

---

---

# CAPI Reference Manual

Version 6.0



## Copyright and Trademarks

*LispWorks CAPI Reference Manual*

Version 6.0

December 2009

Copyright © 2009 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: 617 812 8283  
From elsewhere: +44 870 2206189

[www.lispworks.com](http://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
- active-pane-copy 6
- active-pane-copy-p 6
- active-pane-cut 6
- active-pane-cut-p 6
- active-pane-deselect-all 6
- active-pane-deselect-all-p 6
- active-pane-paste 6
- active-pane-paste-p 6
- active-pane-select-all 6
- active-pane-select-all-p 6
- active-pane-undo 6
- active-pane-undo-p 6

append-items 7  
apply-in-pane-process 8  
arrow-pinboard-object 9  
attach-interface-for-callback 11  
attach-simple-sink 12  
attach-sink 13  
beep-pane 14  
button 15  
button-panel 20  
calculate-constraints 25  
calculate-layout 26  
callbacks 27  
call-editor 29  
capi-object 30  
capi-object-property 31  
check-button 32  
check-button-panel 33  
choice 34  
choice-selected-item 38  
choice-selected-item-p 39  
choice-selected-items 40  
choice-update-item 41  
clipboard 42  
clipboard-empty 44  
clone 45  
cocoa-default-application-interface 45  
cocoa-view-pane 48  
cocoa-view-pane-view 50  
collect-interfaces 51  
collection 52  
collection-find-next-string 55  
collection-find-string 56  
collection-last-search 57  
collection-search 57  
collector-pane 58  
color-screen 59  
column-layout 60  
component-name 63  
confirm-quit 63  
confirm-yes-or-no 65

confirmer-pane 65  
contain 66  
convert-relative-position 67  
convert-to-screen 68  
count-collection-items 72  
current-dialog-handle 72  
current-document 74  
current-pointer-position 74  
current-popup 75  
current-printer 76  
\*default-editor-pane-line-wrap-marker\* 76  
default-library 77  
define-command 77  
define-interface 79  
define-layout 86  
define-ole-control-component 87  
define-menu 88  
destroy 89  
detach-simple-sink 90  
detach-sink 91  
display 92  
display-dialog 94  
display-errors 97  
display-message 97  
display-message-for-pane 98  
display-pane 98  
display-popup-menu 100  
display-replacable-dialog 101  
display-tooltip 102  
docking-layout 103  
docking-layout-pane-docked-p 107  
docking-layout-pane-visible-p 107  
document-container 108  
document-frame 109  
double-headed-arrow-pinboard-object 112  
double-list-panel 113  
drag-pane-object 115  
draw-metafile 117  
draw-metafile-to-image 118  
drawn-pinboard-object 119

draw-pinboard-object 120  
draw-pinboard-object-highlighted 121  
draw-pinboard-object-unhighlighted 121  
drop-object-allows-drop-effect-p 122  
drop-object-collection-index 123  
drop-object-collection-item 124  
drop-object-drop-effect 125  
drop-object-get-object 126  
drop-object-pane-x 127  
drop-object-pane-y 127  
drop-object-provides-format 127  
\*echo-area-cursor-inactive-style\* 128  
echo-area-pane 129  
\*editor-cursor-color\* 129  
\*editor-cursor-active-style\* 130  
\*editor-cursor-drag-style\* 130  
\*editor-cursor-inactive-style\* 130  
editor-pane 131  
editor-pane-blink-rate 140  
editor-pane-buffer 141  
editor-pane-native-blink-rate 142  
editor-pane-selected-text 142  
editor-pane-selected-text-p 143  
editor-pane-stream 143  
editor-window 144  
element 144  
element-container 151  
element-interface-for-callback 151  
element-screen 152  
ellipse 152  
ensure-area-visible 153  
ensure-interface-screen 153  
execute-with-interface 154  
execute-with-interface-if-alive 155  
exit-confirmer 156  
exit-dialog 156  
expandable-item-pinboard-object 157  
extended-selection-tree-view 158  
filtering-layout 158  
filtering-layout-match-object-and-exclude-p 163

find-graph-edge 163  
find-graph-node 164  
find-interface 165  
find-string-in-collection 166  
force-screen-update 166  
force-update-all-screens 167  
foreign-owned-interface 167  
form-layout 168  
free-metafile 169  
free-sound 169  
get-collection-item 170  
get-constraints 170  
get-horizontal-scroll-parameters 171  
get-page-area 172  
get-printer-metrics 173  
get-scroll-position 174  
get-vertical-scroll-parameters 175  
graph-edge 176  
graph-node 176  
graph-node-children 177  
graph-object 177  
graph-pane 178  
graph-pane-add-graph-node 182  
graph-pane-delete-object 183  
graph-pane-delete-objects 183  
graph-pane-delete-selected-objects 184  
graph-pane-direction 184  
graph-pane-edges 185  
graph-pane-nodes 186  
graph-pane-object-at-position 186  
graph-pane-select-graph-nodes 187  
graph-pane-update-moved-objects 187  
grid-layout 188  
hide-interface 193  
hide-pane 193  
highlight-pinboard-object 194  
image-list 195  
image-pinboard-object 196  
image-set 197  
install-postscript-printer 198

- installed-libraries 200
- interactive-pane 201
- interactive-pane-execute-command 203
- interface 204
  - interface-customize-toolbar 220
  - interface-display 221
  - interface-display-title 222
  - interface-editor-pane 223
  - interface-extend-title 223
  - interface-geometry 224
  - interface-iconified-p 225
  - interface-keys-style 225
  - interface-match-p 228
  - interface-menu-groups 229
  - interface-preserve-state 230
  - interface-preserving-state-p 230
  - interface-reuse-p 231
  - interface-toolbar-state 232
  - interface-visible-p 234
- interpret-description 235
- invalidate-pane-constraints 236
- invoke-command 236
- invoke-untranslated-command 237
- item 237
- itemp 239
- item-pinboard-object 239
- labelled-arrow-pinboard-object 240
- labelled-line-pinboard-object 240
- layout 241
- line-pinboard-object 243
- line-pinboard-object-coordinates 244
- list-panel 245
  - list-panel-enabled 252
  - list-panel-filter-state 253
  - list-panel-items-and-filter 254
  - list-panel-unfiltered-items 255
- list-view 256
- listener-pane 261
- listener-pane-insert-value 262
- load-cursor 263

- load-sound 266
- locate-interface 267
- lower-interface 268
- make-container 269
- make-docking-layout-controller 270
- make-foreign-owned-interface 270
- make-general-image-set 272
- make-icon-resource-image-set 273
- make-image-locator 274
- make-menu-for-pane 274
- make-pane-popup-menu 275
- make-resource-image-set 277
- make-scaled-general-image-set 278
- make-scaled-image-set 279
- make-sorting-description 280
- manipulate-pinboard 282
- map-collection-items 285
- map-pane-children 285
- map-pane-descendant-children 288
- map-typeout 289
- \*maximum-moving-objects-to-track-edges\* 289
- menu 290
- menu-component 294
- menu-item 297
- menu-object 303
- merge-menu-bars 306
- message-pane 307
- modify-editor-pane-buffer 308
- mono-screen 309
- move-line 309
- multi-column-list-panel 310
- multi-line-text-input-pane 314
- non-focus-list-interface 315
- non-focus-list-toggle-enable-filter 315
- non-focus-list-toggle-filter 316
- non-focus-list-add-filter 316
- non-focus-list-remove-filter 316
- non-focus-maybe-capture-gesture 316
- non-focus-terminate 318
- non-focus-update 319

prompt-with-list-non-focus 319  
ole-control-add-verbs 324  
ole-control-close-object 325  
ole-control-component 326  
ole-control-doc 328  
ole-control-frame 329  
ole-control-i-dispatch 330  
ole-control-insert-object 331  
ole-control-ole-object 331  
ole-control-pane 332  
ole-control-pane-frame 335  
ole-control-pane-simple-sink 335  
ole-control-user-component 336  
option-pane 337  
output-pane 340  
over-pinboard-object-p 351  
page-setup-dialog 351  
pane-adjusted-offset 352  
pane-adjusted-position 353  
pane-close-display 355  
pane-descendant-child-with-focus 356  
pane-got-focus 356  
pane-has-focus-p 357  
pane-initial-focus 357  
pane-interface-copy-object 359  
pane-interface-copy-p 359  
pane-interface-cut-object 359  
pane-interface-cut-p 359  
pane-interface-deselect-all 359  
pane-interface-deselect-all-p 359  
pane-interface-paste-object 359  
pane-interface-paste-p 359  
pane-interface-select-all 359  
pane-interface-select-all-p 359  
pane-interface-undo 359  
pane-interface-undo-p 359  
pane-popup-menu-items 360  
pane-string 362  
pane-supports-menus-with-images 363  
parse-layout-descriptor 364

password-pane 364  
play-sound 365  
pinboard-layout 366  
pinboard-object 369  
pinboard-object-at-position 372  
pinboard-object-graphics-arg 373  
pinboard-object-overlap-p 374  
pinboard-pane-position 375  
pinboard-pane-size 376  
popup-confirmer 377  
popup-menu-button 388  
print-capi-button 388  
print-collection-item 389  
print-dialog 390  
print-editor-buffer 392  
print-file 392  
print-rich-text-pane 393  
print-text 394  
printer-configuration-dialog 395  
printer-metrics 396  
\*ppd-directory\* 397  
printer-port-handle 397  
printer-port-supports-p 398  
\*printer-search-path\* 399  
process-pending-messages 400  
progress-bar 401  
prompt-for-color 401  
prompt-for-confirmation 402  
prompt-for-directory 404  
prompt-for-file 406  
prompt-for-files 409  
prompt-for-font 410  
prompt-for-form 411  
prompt-for-forms 413  
prompt-for-integer 414  
prompt-for-items-from-list 415  
prompt-for-number 416  
prompt-for-string 417  
prompt-for-symbol 419  
prompt-for-value 421

prompt-with-list 422  
prompt-with-message 424  
push-button 425  
push-button-panel 426  
quit-interface 428  
radio-button 429  
radio-button-panel 430  
raise-interface 431  
range-pane 432  
range-set-sizes 433  
read-sound-file 434  
rectangle 434  
redisplay-collection-item 435  
redisplay-interface 435  
redisplay-menu-bar 436  
redraw-pinboard-layout 437  
redraw-pinboard-object 437  
reinitialize-interface 438  
remove-capi-object-property 438  
remove-items 439  
replace-dialog 440  
replace-items 441  
report-active-component-failure 442  
reuse-interfaces-p 443  
rich-text-pane 444  
rich-text-pane-character-format 446  
rich-text-pane-operation 447  
rich-text-pane-paragraph-format 451  
rich-text-version 452  
right-angle-line-pinboard-object 452  
row-layout 453  
screen 455  
screen-active-interface 457  
screen-active-p 457  
screen-logical-resolution 458  
screen-internal-geometry 458  
screens 459  
scroll 460  
scroll-bar 462  
scroll-if-not-visible-p 463

search-for-item 465  
selection 465  
selection-empty 466  
set-application-interface 467  
set-button-panel-enabled-items 468  
set-clipboard 468  
set-confirm-quit-flag 470  
set-default-editor-pane-blink-rate 470  
set-default-interface-prefix-suffix 471  
set-drop-object-supported-formats 473  
set-editor-parenthesis-colors 474  
set-geometric-hint 475  
set-hint-table 475  
set-horizontal-scroll-parameters 476  
set-interactive-break-gestures 477  
set-object-automatic-resize 478  
set-pane-focus 482  
set-rich-text-pane-character-format 482  
set-rich-text-pane-paragraph-format 485  
set-selection 487  
set-printer-metrics 488  
set-printer-options 489  
set-text-input-pane-selection 490  
set-top-level-interface-geometry 491  
set-vertical-scroll-parameters 492  
shell-pane 493  
show-interface 495  
show-pane 495  
simple-layout 496  
simple-network-pane 496  
simple-pane 497  
simple-pane-handle 505  
simple-pane-visible-height 506  
simple-pane-visible-size 506  
simple-pane-visible-width 507  
simple-pinboard-layout 508  
simple-print-port 509  
slider 510  
sort-object-items-by 511  
sorted-object 512

sorted-object-sort-by 512  
start-gc-monitor 513  
static-layout 514  
stop-gc-monitor 515  
stop-sound 516  
switchable-layout 516  
switchable-layout-switchable-children 518  
tab-layout 518  
tab-layout-panes 521  
tab-layout-visible-child 522  
text-input-choice 522  
text-input-pane 523  
text-input-pane-complete-text 536  
text-input-pane-copy 537  
text-input-pane-cut 538  
text-input-pane-delete 538  
text-input-pane-in-place-complete 539  
text-input-pane-paste 539  
text-input-pane-selected-text 540  
text-input-pane-selection 540  
text-input-pane-selection-p 541  
text-input-range 542  
title-pane 543  
titled-menu-object 544  
titled-object 546  
titled-pinboard-object 549  
toolbar 552  
toolbar-button 554  
toolbar-component 559  
toolbar-object 561  
top-level-interface 561  
top-level-interface-display-state 562  
top-level-interface-geometry 563  
top-level-interface-geometry-key 564  
top-level-interface-p 566  
top-level-interface-save-geometry-p 566  
tracking-pinboard-layout 567  
tree-view 569  
tree-view-ensure-visible 578  
tree-view-expanded-p 578

- tree-view-item-checkbox-status 579
- tree-view-item-children-checkbox-status 580
- tree-view-update-an-item 580
- tree-view-update-item 581
- undefine-menu 582
- unhighlight-pinboard-object 582
- uninstall-postscript-printer 582
- unmap-typeout 583
- update-all-interface-titles 584
- update-interface-title 584
- update-pinboard-object 585
- update-screen-interface-titles 585
- \*update-screen-interfaces-hooks\* 586
- update-toolbar 586
  - with-atomic-redisplay 587
  - with-busy-interface 587
  - with-dialog-results 588
  - with-document-pages 591
  - with-external-metafile 592
  - with-geometry 594
  - with-internal-metafile 596
  - with-output-to-printer 597
  - with-page 598
  - with-page-transform 598
  - with-print-job 599
  - with-random-typeout 600
- wrap-text 600
- wrap-text-for-pane 601
- x-y-adjustable-layout 602

## **2 GP Reference Entries 605**

- analyze-external-image 605
- apply-rotation 606
- apply-scale 606
- apply-translation 607
- augment-font-description 607
- clear-external-image-conversions 608
- clear-graphics-port 609
- clear-graphics-port-state 609

clear-rectangle 610  
compress-external-image 610  
compute-char-extents 611  
convert-external-image 611  
convert-to-font-description 612  
copy-external-image 613  
copy-pixels 613  
copy-transform 614  
create-pixmap-port 615  
\*default-image-translation-table\* 616  
define-font-alias 616  
destroy-pixmap-port 617  
dither-color-spec 617  
draw-arc 618  
draw-arcs 619  
draw-character 620  
draw-circle 620  
draw-ellipse 621  
draw-image 622  
draw-line 623  
draw-lines 624  
draw-point 625  
draw-points 625  
draw-polygon 626  
draw-polygons 627  
draw-rectangle 628  
draw-rectangles 629  
draw-string 630  
ensure-gdiplus 631  
external-image 632  
external-image-color-table 633  
external-image-color-table 633  
externalize-image 634  
find-best-font 635  
find-matching-fonts 636  
font-description 636  
font-description-attributes 637  
font-description-attribute-value 637  
font-fixed-width-p 638  
free-image 638

free-image-access 639  
get-bounds 639  
get-character-extent 640  
get-char-ascent 641  
get-char-descent 642  
get-char-width 642  
get-enclosing-rectangle 643  
get-font-ascent 643  
get-font-average-width 644  
get-font-descent 644  
get-font-height 645  
get-font-width 645  
get-graphics-state 646  
get-origin 646  
get-string-extent 647  
get-transform-scale 648  
graphics-port-transform 648  
image 649  
image-access-pixel 649  
image-access-pixels-from-bgra 650  
image-access-pixels-to-bgra 651  
image-access-transfer-from-image 652  
image-access-transfer-to-image 653  
image-freed-p 654  
image-loader 654  
image-translation 655  
initialize-dithers 656  
inset-rectangle 656  
inside-rectangle 657  
invalidate-rectangle 658  
invert-transform 658  
list-all-font-names 659  
load-icon-image 659  
load-image 661  
make-dither 664  
make-font-description 664  
make-graphics-state 665  
make-image 670  
make-image-access 671  
make-image-from-port 672

make-sub-image 673  
make-transform 673  
merge-font-descriptions 674  
offset-rectangle 675  
ordered-rectangle-union 675  
pixmap 676  
pixmap-port 677  
port-height 678  
port-string-height 678  
port-string-width 679  
port-width 679  
postmultiply-transforms 680  
premultiply-transforms 680  
read-and-convert-external-image 681  
read-external-image 682  
rectangle-bind 683  
rectangle-bottom 684  
rectangle-height 684  
rectangle-left 685  
rectangle-right 685  
rectangle-top 686  
rectangle-union 686  
rectangle-width 687  
rect-bind 688  
register-image-load-function 688  
register-image-translation 689  
reset-image-translation-table 690  
separation 691  
set-default-image-load-function 691  
set-graphics-port-coordinates 692  
set-graphics-state 693  
transform 693  
transform-area 694  
transform-distance 694  
transform-distances 695  
transform-is-rotated 695  
transform-point 696  
transform-points 696  
transform-rect 697  
undefine-font-alias 698

- union-rectangle 698
- \*unit-transform\* 699
- unit-transform-p 699
- unless-empty-rect-bind 700
- untransform-distance 700
- untransform-distances 701
- untransform-point 701
- untransform-points 702
- validate-rectangle 703
- with-dither 704
- with-graphics-mask 704
- with-graphics-rotation 705
- with-graphics-scale 706
- with-graphics-state 707
- with-graphics-transform 708
- with-graphics-translation 709
- with-inverse-graphics 709
- without-relative-drawing 710
- with-pixmap-graphics-port 710
- with-transformed-area 711
- with-transformed-point 712
- with-transformed-points 713
- with-transformed-rect 714
- write-external-image 714

### **3 COLOR Reference Entries 717**

- apropos-color-alias-names 717
- apropos-color-names 718
- apropos-color-spec-names 719
- color-alpha 720
- color-*<component>* 720
- \*color-database\* 721
- color-level 722
- color-model 723
- colors= 723
- convert-color 724
- define-color-alias 725
- define-color-models 726
- delete-color-translation 727

ensure-*<command>* 728  
get-all-color-names 729  
get-color-alias-translation 730  
get-color-spec 731  
load-color-database 732  
make-gray 733  
make-hsv 734  
make-rgb 735  
read-color-db 736  
unconvert-color 737

**Index** 739

---

---

# 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”, “Method Signature”, “Initial Value”, “Superclasses”, “Subclasses”, “Initargs”, “Accessors”, “Readers”, “Compatibility Note”, “Description”, “Notes”, “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, slots and return values 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 and Reference Manual* describes the main language-level features and tools available in LispWorks, along with reference pages.
- The *LispWorks IDE User Guide* describes the LispWorks IDE, the user interface for LispWorks. This is a set of windowing tools that help you to develop and test Common Lisp programs.

- The *LispWorks Editor User Guide* describes the keyboard commands and programming interface to the LispWorks IDE 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 *LispWorks IDE 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](mailto: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.

<b>abort-callback</b>	<i>Function</i>
Summary	Aborts out of the context of the current callback.
Package	<code>capi</code>
Signature	<code>abort-callback &amp;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 on 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`

**active-pane-copy**  
**active-pane-copy-p**  
**active-pane-cut**  
**active-pane-cut-p**  
**active-pane-deselect-all**  
**active-pane-deselect-all-p**  
**active-pane-paste**  
**active-pane-paste-p**  
**active-pane-select-all**  
**active-pane-select-all-p**  
**active-pane-undo**  
**active-pane-undo-p**

*Functions*

Summary      Perform, or check applicability of, an "edit/select operation"  
on the active pane.

Signature      `active-pane-copy` &optional *pane*  
`active-pane-copy-p` &optional *pane*  
`active-pane-cut` &optional *pane*  
`active-pane-cut-p` &optional *pane*  
`active-pane-deselect-all` &optional *pane*  
`active-pane-deselect-all-p` &optional *pane*  
`active-pane-paste` &optional *pane*  
`active-pane-paste-p` &optional *pane*  
`active-pane-select-all` &optional *pane*  
`active-pane-select-all-p` &optional *pane*  
`active-pane-undo` &optional *pane*  
`active-pane-undo-p` &optional *pane*

**Description** These functions perform an "edit/select operation" on the active pane, or check if this operation is currently applicable. The active pane will be the one on the same screen as *pane* if *pane* is non-nil, or otherwise the same screen as the default interface.

These functions find the active pane, that is the pane where keyboard input currently goes. Note that this is not necessarily a pane that is recognized by CAPI. The predicates (those with names ending `-p`) return true if the operation is currently applicable. The other functions tell the active pane to do the operation.

The edit/select operations are implemented by the `pane-interface-*` generic functions such as `pane-interface-copy-object`.

It is not an error to do the operation even if the predicate returns false. It will just do nothing useful.

**Examples** See `examples/capi/applications/rich-text-editor.lisp`

**See also** `pane-interface-copy-object`

## append-items

*Generic Function*

**Summary** Adds to the items in a collection.

**Signature** `append-items collection new-items`

**Arguments** *collection* A collection.  
*new-items* A sequence.

**Description** The generic function `append-items` adds the items in *new-items* to the *collection collection*.

This is logically equivalent to recalculating the collection items and calling `(setf collection-items)`. However, `append-items` is more efficient and causes less flickering on screen.

`append-items` can only be used when the `collection` has the default *items-get-function* `svref`.

See also `collection`  
`remove-items`  
`replace-items`

## apply-in-pane-process

*Function*

Summary Applies a function in the process associated with a pane.

Package `capi`

Signature `apply-in-pane-process pane function &rest args => nil`

Description The function `apply-in-pane-process` applies *function* to *args* in the process that is associated with *pane*. This is required when *function* modifies *pane* or changes how it is displayed. If *pane* has not been displayed yet, then *function* is called immediately.

- Notes
1. All accesses (reads as well as writes) on a pane should be performed in the pane's process. Within a callback on the pane's interface this happens automatically, but `apply-in-pane-process` is a useful utility in other circumstances.
  2. `apply-in-pane-process` calls *function* on the current process if the pane's interface does not have a process.

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	<p><code>:head</code> A keyword specifying the position of the arrowhead on the line.</p> <p><code>:head-direction</code> A keyword specifying the direction of the arrowhead.</p> <p><code>:head-length</code> The length of the arrowhead.</p> <p><code>:head-breadth</code> The breadth of the arrowhead, or <code>nil</code>.</p> <p><code>:head-graphics-args</code> A graphics args plist.</p>
Description	<p>An instance of the class <code>arrow-pinboard-object</code> is a <code>pinboard-object</code> that draws itself as an arrow.</p> <p><i>head</i> must be <code>:end</code>, <code>:middle</code> or <code>:start</code>. The default is <code>:end</code>.</p>

*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 `make-graphics-state`.

## 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	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	The function <code>attach-interface-for-callback</code> changes the interface that is passed when a callback is made. Callbacks for <i>element</i> get passed <i>interface</i> instead of <i>element</i> 's 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 &amp;key sink-class =&gt; 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 `ole-control-pane`.

Package `capi`

Signature `attach-sink sink pane interface-name`

Arguments *sink* A class instance.  
*pane* An `ole-control-pane`.  
*interface-name* A refguid or the symbol `:default`.

Description The function `attach-sink` attaches a sink to the active component in the the `ole-control-pane` *pane*.

*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	Sounds a beep.
Package	<code>capi</code>
Signature	<code>beep-pane &amp;optional pane</code>
Description	The function <code>beep-pane</code> 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 two initargs controlling alternate images apply only on Motif:	
	<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.

The following initarg controlling another alternate image applies only on GTK+ and Motif:

`: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 some platforms 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-panel` provides functionality to group buttons together, and should normally be used in preference to creating individual buttons yourself. For instance, a `radio-button-panel` 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`.

Notes            The `simple-pane` initarg *foreground* is not supported for buttons on Windows and Cocoa.

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 `button-panel` is a pane containing a number of buttons that are laid out in a particular style, and that have group behavior.

**Package**      `capi`

**Superclasses**    `choice`  
                   `titled-object`  
                   `simple-pane`

**Subclasses**     `push-button-panel`  
                   `radio-button-panel`  
                   `check-button-panel`

**Initargs**        `:layout-class` The type of layout for the buttons.  
                   `:layout-args` Initialization arguments for the layout.  
                   `:callbacks`    The selection callbacks for each button.  
                   `:button-class` The class of the buttons.  
                   `:images`        A list.  
                   `:disabled-images`  
                                   A list.  
                   `:armed-images` A list.

`:selected-images`  
A list.

`:selected-disabled-images`  
A list.

`:help-keys` A list.

`:default-button`  
Specifies the default button.

`:cancel-button`  
Specifies the cancel button.

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	Calculates the minimum and maximum size of a pane.
Package	<code>capi</code>
Signature	<code>calculate-constraints <i>pane</i></code>
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.</p>

`%max-width%` The maximum width of pane.

`%min-height%` The minimum height of pane.

`%max-height%` The maximum height of pane.

(See `with-geometry`.)

The constraints of any CAPI element can be found by calling `get-constraints`.

See also

- `calculate-layout`
- `define-layout`
- `get-constraints`
- `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 `capi-object`

Subclasses `collection`  
`item`  
`menu-object`

Initargs `:callback-type` The type of arguments for the callbacks.

`:selection-callback`

The callback for selecting an item.

`:extend-callback`

The callback for extending the selection.

`:retract-callback`

The callback for deselecting an item.

`:action-callback`

The callback for an action.

Accessors `callbacks-callback-type`  
`callbacks-selection-callback`  
`callbacks-extend-callback`  
`callbacks-retract-callback`  
`callbacks-action-callback`

Description Each callback function can be one of the following:

*function*            Call the function.

*list*                Apply the head of the list to the tail.

`:redisplay-interface`  
                      Call `redisplay-interface` on the top-level interface.

`:redisplay-menu-bar`  
                      Call `redisplay-menu-bar` on the top-level interface.

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:

`:collection-data`  
                      (*collection data*)

`:data`                (*item-data*)

`:data-element`      (*item-data element*)

`:data-interface`  
                      (*item-data interface*)

`:element`            (*element*)

`:element-data`      (*element item-data*)

`:element-item`      (*element item*)

`:interface-data`  
                      (*interface item-data*)

`:item`                (*item*)

`:item-element`      (*item element*)

`:item-interface`  
                      (*item interface*)

`:interface-item`                   (*interface item*)

`:interface`                   (*interface*)

`:full`                   (*item-data item interface*)

`:focus`               The pane with the current input focus.

`:none`               ()

`nil`               ()

*callback-type* can also be a list containing any of `:focus`, `:data`, `:element`, `: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. The *element* variable means the element containing the callback itself.

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

## **call-editor**

## *Generic Function*

Summary           Executes an editor command in an `editor-pane`.

Package           `capi`

Signature         `call-editor editor-pane command`

Description       The generic function `call-editor` 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 `capi-object` is the superclass of all CAPI classes.

**Package** `capi`

**Superclasses** `standard-class`

**Subclasses** `item`  
`callbacks`  
`element`  
`interface`  
`pinboard-object`

**Initargs** `:name` The name of the object.

`:plist` A property list for storing miscellaneous information.

Accessors `capi-object-name`  
`capi-object-plist`

Description The class `capi-object` provides a name and a property list for general purposes, along with the accessors `capi-object-name` and `capi-object-plist` respectively. A `capi-object`'s name is defaulted by `define-interface` to be the name of the slot into which the object is put.

Example 

```
(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)
```

See also `capi-object-property`

## **capi-object-property**

*Function*

Summary The `capi-object-property` function is used to get and set properties in the property list of a `capi-object`.

Package `capi`

Signature `capi-object-property` *object property*

Signature `(setf capi-object-property)` *value object property*

**Description** All CAPI objects contain a property list, similar to the symbol `plist`. The recommended ways of setting properties are `capi-object-property` and `(setf capi-object-property)`. To remove a property, use the function `remove-capi-object-property`.

**Example** In this example a list panel is created, and a test property is set and examined using `capi-object-property`.

```
(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`  
`remove-capi-object-property`

## 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 your-

self, 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 `check-button-panel` is a pane containing a group of buttons each of which can be selected or deselected.

Package      `capi`

Superclasses `button-panel`

**Description**      The class `check-button-panel` inherits all of its behavior from `button-panel`, which itself inherits most of its behavior from `choice`. Thus, the `check-button-panel` can accept *items*, *callbacks*, and so on.

**Example**

```
(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)
```

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

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

## choice

*Class*

**Summary**      A `choice` is an abstract class that collects together a group of items, and provides functionality for displaying and selecting them.

**Package**      `capi`

**Superclasses** `collection`

Subclasses	<p> <code>button-panel</code>  <code>extended-selection-tree-view</code>  <code>graph-pane</code>  <code>list-panel</code>  <code>menu-component</code>  <code>option-pane</code>  <code>tree-view</code> </p>
Initargs	<p> <code>:interaction</code>    The interaction style of the choice.  <code>:selection</code>        The indexes of the choice's selected items.  <code>:selected-item</code>                        The selected item for a single selection                        choice.  <code>:selected-items</code>                        A list of the selected items.  <code>:keep-selection-p</code>                        If <code>t</code>, retains any selection when the items                        change.  <code>:initial-focus-item</code>                        If supplied, this should be an item in the                        choice. </p>
Accessors	<p><code>choice-selection</code></p>
Readers	<p> <code>choice-interaction</code>  <code>choice-initial-focus-item</code> </p>
Description	<p>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.</p> <p>A <code>choice</code> can have one of four different interaction styles, and these control how it behaves when an item is selected by the user. <i>interaction</i> can be one of:</p>

`: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) =&gt; t  Now click on another item.  (capi:choice-selected-item-p list 'c) =&gt; 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**     The function `choice-selected-items` returns the currently selected items in a choice as a list of the items. A `setf` method is provided as a means of setting the currently selected items. Note that the items are compared by *choice's test-function* - see `collection` for details.

In the case of `:single-selection` choices, it is usually easier to use the complementary function `choice-selected-item`, which returns the selected item as its result.

**Example**            First we set up a `:multiple-selection` choice — in this case, a list panel.

```
(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))))
```

The following code line returns the selections of the list.

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

The selections of the list panel can be changed and redisplayed using the following code.

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

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

**See also**            `choice`  
                      `choice-selected-item`  
                      `collection`

## **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 &amp;optional format =&gt; result</code>
Arguments	<p><i>self</i>            A displayed CAPI pane or interface.</p> <p><i>format</i>         A keyword.</p>
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> <p><code>:string</code>         The object is a string. This the default value.</p> <p><code>:image</code>         The object is of type <code>image</code>, converted from whatever format the platform supports.</p> <p><code>:value</code>         The object is the Lisp value.</p> <p><code>:metafile</code>      The object is a metafile.</p> <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 GTK+ or 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 the Lisp-Works editor 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</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>	<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 &amp;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 *create-callback* is called when the application starts up.
- The *destroy-callback* is called when the application shuts down.
- The *confirm-destroy-function* is called to confirm whether the application should shut down.

All of these callbacks execute in the thread that runs the Cocoa event loop, so they can call CAPI and GP functions.

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

`:others-hidden`

Hide Others

`:all-normal` 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.

Example

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 to 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.

Example         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 <code>cocoa-view-pane</code> .
Package	<code>capi</code>
Signature	<code>cocoa-view-pane-view <i>pane</i> =&gt; <i>view</i></code>
Arguments	<i>pane</i> A <code>cocoa-view-pane</code> .
Values	<i>view</i> A foreign pointer to a Cocoa view or <code>nil</code> .
Description	<p>The function <code>cocoa-view-pane-view</code> returns the Cocoa view for the <code>cocoa-view-pane</code> <i>pane</i> as a foreign pointer. This view is only accessible when the pane is visible and <code>nil</code> is returned in other cases.</p> <p><b>Note:</b> <code>cocoa-view-pane-view</code> is implemented only in LispWorks for Macintosh with the Cocoa IDE. See the <i>LispWorks Objective-C and Cocoa Interface User Guide and Reference Manual</i> for details on using Cocoa.</p>
Example	See the example in <code>examples/objc/movie-view.lisp</code> .
See also	<code>cocoa-view-pane</code>

## collect-interfaces

## Generic Function

Summary	Finds all interfaces of a given class.	
Package	<code>capi</code>	
Signature	<code>collect-interfaces proto &amp;key screen current-process-first sort-by =&gt; 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`.

#### Example

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 &amp;key set string =&gt; 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> &amp;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*`.

### Example

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 keywords `:divider` and `:separator` which automatically create a divider or separator as a child of the `column-layout`. The user can move a divider, but cannot move a separator.

When specifying `:ratios` in a row with `:divider` or `:separator`, you should use `nil` to specify that the divider or separator is given its minimum size, as in the example below.

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

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))))

```

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> =&gt; <i>name</i></code> <code>(setf component-name) <i>name pane</i> =&gt; <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><b>Note:</b> <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 > Preferences... > Environment > General > 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 <code>confirm-yes-or-no</code> pops up a dialog button containing a message and a <b>Yes</b> and <b>No</b> button.
Package	<code>capi</code>
Signature	<code>confirm-yes-or-no <i>format-string</i> &amp;rest <i>format-args</i></code>
Description	<p>This pops up a dialog box containing a message and the buttons <b>Yes</b> and <b>No</b>, returns <code>t</code> when the <b>Yes</b> button is clicked, and <code>nil</code> when the <b>No</b> 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	<code>prompt-for-confirmation</code> <code>display-dialog</code> <code>popup-confirmer</code>

## confirmer-pane

*Function*

Summary	Returns the pane associated with a confirmer interface.
Package	<code>capi</code>

Signature	<code>confirmer-pane</code> <i>interface</i> => <i>pane</i>
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</code> <i>element</i> &rest <i>interface-args</i> &key <i>screen process title</i> &allow-other-keys => <i>element</i>
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>

```
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`.

*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	<code>convert-relative-position</code> <i>from to</i> <i>x y</i> => <i>to-x, to-y</i>	
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</code> &optional <i>object</i> => <i>result</i>	
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 *object*.

If *object* is `nil`, *result* is the default screen. *object* defaults to `nil`.

If *object* is a pane inside a MDI interface, then *result* is the `capl: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 keyword representing the CAPI library. This is equivalent to using the `:library` key in the plist case below.

*object* can be a plist. The keys below are supported on GTK+ and 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 (on Motif) the `-display` command-line option, or (on GTK+) 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.

On GTK+ the value is used for constructing the default *widget-name* for top-level interfaces. The application-class is prepended to the interface name followed by a ".", so if *application-class* is "my-application", a top-level-interface of class `my-interface` will have a default *widget-name* "my-application.my-interface".

See `element` for the description of *widget-name*.

Example GTK+ resource files are in `lib/6-0-0-0/examples/gtk/`

**:fallback-resources**

On GTK+ the fallback resources are global, so they cannot be used to define different resources for different screens. Each call to `convert-to-screen` where *fallback-resources* is passed overrides the previous call. The value of *fallback-resources* is either a single string or a list of strings. In either case each string must be a complete specification according to the standard resource specifica-

tion of GTK+ resource files (`gtk_rc_parse_string` should be able to parse it).

On Motif 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.

`:library` The value specifies the CAPI library. This is useful on Linux, FreeBSD and x86/x64 Solaris platforms, and in the Mac OS X/GTK+ image, to choose between `:gtk` and `:motif` if the deprecated "capi-motif" module is loaded.

This key is supported on Motif only. Other libraries ignore it.

`: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.

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 `count-collection-items` *collection* &optional *representation*

Description The `count-collection-items` generic function returns the number of items in *collection* by calling the *items-count-function*.

*representation* defaults to `nil`. If it is non-`nil`, it is used instead of the *items* of *collection*.

Example The following example uses `count-collection-items` to return the number of items in a list panel.

```
(setq list (make-instance 'capi:list-panel  
                        :items '(1 2 3 4 5)))
```

```
(capi:count-collection-items list)
```

The following example shows how to count the number of items in a specified list.

```
(capi:count-collection-items list '(1 2))
```

See also `collection`  
`get-collection-item`  
`search-for-item`

## current-dialog-handle

*Function*

Summary Returns the underlying handle of the current dialog.

Package	<code>capi</code>
Signature	<code>current-dialog-handle =&gt; <i>handle</i></code>
Values	<i>handle</i> A platform-specific value, or <code>nil</code> .
Description	<p>The function <code>current-dialog-handle</code> returns the underlying handle of the current dialog, as follows:</p> <p>Microsoft Windows</p> <p style="padding-left: 40px;">The <code>hwnd</code> of the dialog.</p> <p>GTK+</p> <p style="padding-left: 40px;">A pointer to the <code>GdkWindow</code>.</p> <p>Motif</p> <p style="padding-left: 40px;">A <code>windowid</code> of the dialog.</p> <p>Cocoa</p> <p style="padding-left: 40px;">The value returned by the <code>NSWindow</code>'s <code>windowNumber</code> method.</p> <p>This value is useful if you want to perform some operation on the underlying handle that the CAPI does not supply.</p> <p>If there is no current dialog, <code>current-dialog-handle</code> returns <code>nil</code>.</p>
Example	<p>Press on "Get handle" to see the handle of the dialog.</p> <pre>(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")</pre>
See also	<code>simple-pane-handle</code>

**current-document***Generic Function*

Summary	Returns the current document of a MDI interface.	
Package	<code>capi</code>	
Signature	<code>current-document <i>mdi-interface</i> =&gt; <i>child</i></code>	
Arguments	<i>mdi-interface</i>	An instance of a subclass of <code>document-frame</code> .
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 &amp;key <i>relative-to</i> <i>pane-relative-p</i> =&gt; <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     The function `current-pointer-position` returns the current x,y position of the pointer on the screen of *relative-to*, which defaults to the current screen.

If *pane-relative-p* is true then the position is returned relative to *relative-to*, otherwise it is returned relative to the screen. The default value of *pane-relative-p* is `t`.

See also         `interface`  
                  `screen`

## current-popup

*Function*

Summary         Returns the current popup pane if there is one.

Signature        `current-popup => result`

Values           *result*            A pane or `nil`.

Description     The function `current-popup` returns the current popup pane or `nil` if there is none. A current popup exists in the scope of callbacks which are done while a dialog is displayed on the screen in the current process.

If the dialog was raised by an explicit call to `display-dialog` or `popup-confirmer`, `current-popup` returns the first argument of `display-dialog` or `popup-confirmer`. For other functions that raise a dialog (such as the `prompt-for-file`, `prompt-for-confirmation` and so on), the result is CAPI pane created by the system.

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

**current-printer***Function*

Summary	Returns the currently selected printer object.
Package	<code>capi</code>
Signature	<code>current-printer &amp;key <i>interactive</i> =&gt; <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

*Function*

Summary	Returns the default library.
Package	<code>capi</code>
Signature	<code>default-library =&gt; <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 Linux, FreeBSD and xw86/x64 Solaris platforms, the default library is <code>:gtk</code>. If you load the deprecated "capi-motif" module, then the library will be <code>:motif</code>.</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, the only library available in the native GUI image is <code>:cocoa</code>, hence this is the default library. In the Mac OS X/GTK+ image, the default library is <code>:gtk</code>, but you load the deprecated "capi-motif" module, then the library will be <code>:motif</code>.</p> <p>In LispWorks for UNIX only (not LispWorks for Linux, FreeBSD, or x86/x64 Solaris) platforms, currently the only library available is <code>:motif</code>, hence this is the default library.</p>
See also	<code>installed-libraries</code>

## define-command

*Macro*

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</i> <i>gesture</i> &amp;<i>key</i> <i>translator</i> <i>host</i></code>
Description	<p>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 <i>translator</i> 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 <code>output-pane</code> (which will be the first argument). The <i>host</i> indicates which platforms this gesture should apply for (it defaults to all platforms).</p> <p>For a full description of the gesture syntax, see <code>output-pane</code>.</p>

Example 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 `define-interface` macro defines subclasses of  
**interface**.

**Package**     `capi`

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

Description The macro `define-interface` is used to define subclasses of `interface`, which when created with `make-instance` has the specified panes, layouts and menus created automatically. If non-nil, *superclasses* must include `interface` or a subclass of it.

`define-interface` is essentially a version of `defclass` 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 `capi: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.

Example

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 `define-menu` macro defines a menu function.

Package      `capi`

Signature     `define-menu function-name (self) title  
   menu-body &rest menu-options`

Description    The macro `define-menu` defines a function called *function-name* with a single argument *self* that will make a menu. The parameters *title*, *menu-body* and *menu-options* take the same form as the `:menus` section of `define-interface`.

```

Example  (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)

```

See also `define-interface`  
`menu`

## destroy

## Generic Function

Summary Closes a window and calls the *destroy-callback*.

Package `capi`

Signature `destroy interface`

Description The generic function `destroy` closes the window associated with *interface*, and then calls the interface's *destroy-callback* if it has one.

There is a complementary function `quit-interface` which calls the interface's *confirm-destroy-function* 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 *confirm-destroy-function* is ignored.

**Note:** `destroy` must only be called in the process of *interface*. Menu callbacks on *interface* will be called in that process, but otherwise you probably need to use `execute-with-interface` or `apply-in-pane-process`.

Example

```
(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)
```

See also

```
interface
quit-interface
*update-screen-interfaces-hooks*
```

## detach-simple-sink

*Function*

Summary Detaches a previously-attached simple sink object.

Package `capi`

Signature `detach-simple-sink sink pane`

Arguments *sink* A class instance.  
*pane* An `ole-control-pane`.

Description The function `detach-simple-sink` detaches a sink that was previously attached to the active component in the `ole-control-pane` *pane* by a call to `attach-simple-sink`.

*sink* is the value returned by `attach-simple-sink` when the sink was attached.

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

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-simple-sink`  
`ole-control-pane`

## detach-sink

*Function*

Summary Detaches a previously-attached sink.

Package `capi`

Signature `detach-sink sink pane interface-name`

Arguments *sink* A class instance.  
*pane* An `ole-control-pane`.  
*interface-name* A refguid or the symbol `:default`.

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 Displays a CAPI interface on a specified screen.

Package `capi`

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

Arguments

<i>interface</i>	A CAPI interface.
<i>screen</i>	A screen, or any argument accepted by <code>convert-to-screen</code> .
<i>owner</i>	A CAPI interface.
<i>window-styles</i>	A list of keywords.
<i>process</i>	On GTK+, Windows or Motif, a CAPI process, <code>t</code> or <code>nil</code> . On Cocoa, this argument is not supported.

Values *interface* A CAPI interface.

Description The function `display` displays the CAPI interface *interface* on the specified *screen* (or the current one if not supplied).

If *process* is not supplied, then if *owner* is supplied *interface* runs in *owner*'s process, otherwise *interface* runs in the process of the parent of *interface* if it is a `document-container`, or in a new process created for *interface* if not.

On Windows and Motif, if *process* is `t`, then *interface* runs in a newly-created process. If *process* is `nil`, *interface* runs in the current process. Otherwise *process* is expected to be a CAPI process, and *interface* runs in it. A CAPI process is a `mp:process` which was created by calling `display`. You can pass only a CAPI process as *process*, because it needs to handle messages using the LispWorks event loop. The default value of *process* is `t`.

On Cocoa, all CAPI interfaces run in the Cocoa Event Loop process (which is the main thread of LispWorks) and therefore the *process* argument is not supported. If the value of *process* is any process other than the Cocoa Event Loop process an error is signalled.

*owner* specifies an owner for *interface*, which should be another CAPI interface. *interface* inherits a number of attributes from *owner*, including the default process, default screen and default display state.

*window-styles*, if supplied, sets the *window-styles* slot of *interface*. See `interface` for information about *window-styles*.

`display` returns its *interface* argument.

**Note:** Use the function `contain` to display objects other than interfaces.

**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 <code>display-dialog</code> function displays a CAPI interface as a dialog box.	
Package	<code>capi</code>	
Signature	<code>display-dialog</code> <i>interface</i> &key <i>screen focus modal owner x y position-relative-to continuation callback-error-handler</i> => <i>result, okp</i>	
Arguments	<i>interface</i>	A CAPI interface.
	<i>screen</i>	A screen.
	<i>focus</i>	A pane of <i>interface</i> .
	<i>modal</i>	<code>t</code> , <code>:dismiss-on-input</code> OR <code>nil</code> .
	<i>owner</i>	A pane.
	<i>x, y</i>	Real numbers representing coordinates, or keywords or lists specifying an adjusted position.
	<i>position-relative-to</i>	<code>:owner</code> OR <code>nil</code> .
	<i>continuation</i>	A function or <code>nil</code> .
	<i>callback-error-handler</i>	A function designator or <code>nil</code> .
Values	<i>result</i>	An object.
	<i>okp</i>	A boolean.

Description This is a complementary function that displays the CAPI `interface` *interface* as a dialog box.

`screen` is the `screen` for the dialog to be displayed on.

`focus` 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.

A true value of `modal` indicates that the dialog takes over all input to the application. Additionally, if `modal` is `:dismiss-on-input` then any user gesture (a button or key press) causes the dialog to disappear. `:dismiss-on-input` works on platforms other than Motif. The default value of `modal` is `t`.

`owner` specifies an owner window for the dialog. See the "Prompting for Input" chapter in the *LispWorks CAPI User Guide* for details.

If `x` and `y` are numbers they specify the coordinates of the dialog. Alternatively `x` and `y` can be keywords like `:left` and `:top`, or lists like `(:left 100)`, `(:bottom 50)` and so on.. These values cause the dialog to be positioned relative to its owner in the same way as the `adjust` argument to `pane-adjusted-position`. The default location is at the center of the dialog's owner.

`position-relative-to` has a default value `:owner`, meaning that `x` and `y` are relative to dialog's owner. The value `nil` means that `x` and `y` are relative to the screen.

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.

`callback-error-handler` allows error handling in callbacks which is uniform across platforms, as described for `popup-confirmer`.

#### Notes

1. If you need to replace one dialog with another, you can use `display-replacable-dialog` and `replace-dialog`.
2. In a modal dialog at least one button which aborts or exits the dialog must be provided in `interface`. This is the programmer's responsibility, as without such a button there is no way to clear the modal dialog. A straightforward way to add these buttons is to display the window via `popup-confirmer` which adds the buttons for you.

#### 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 &amp;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> &amp;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`.

**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`.

**Example**

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

(capi:display-message-for-pane pane
  "Just created ~S" 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

<i>menu</i>	A menu.
<i>owner</i>	A pane.
<i>x</i>	The horizontal coordinate of <i>menu</i> 's position relative to <i>owner</i> .
<i>y</i>	The vertical coordinate of <i>menu</i> 's position relative to <i>owner</i> .
<i>button</i>	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	<code>capi</code>	
Signature	<code>display-replacable-dialog</code> <i>interface</i> &rest <i>args</i> => <i>result</i>	
Arguments	<i>interface</i>	An interface.
	<i>args</i>	Other arguments as for <code>display-dialog</code> .
Values	<i>result</i>	The value returned by the dialog.
Description	The function <code>display-replacable-dialog</code> 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> &amp;key <i>x y text</i> =&gt; <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`.

Example 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 pane</i> &key <i>anywhere</i> => <i>dockedp</i>
Signature	<code>(setf docking-layout-pane-docked-p)</code> <i>dockedp docking-layout pane</i> => <i>dockedp</i>
Arguments	<i>docking-layout</i> An instance of <code>docking-layout</code> or a subclass. <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 pane</i> => <i>visiblep</i>

Signature	<code>(setf docking-layout-pane-visible-p) <i>visiblep</i> <i>docking-layout</i> <i>pane</i> =&gt; <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	<b>drag-pane-object</b> <i>pane value &amp;key string plist image-function operations =&gt; 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. The image that is returned by *image-function* is freed automatically in the end of dragging operation. It must be a new image, and cannot be reused.

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

<i>pane</i>	An <code>output-pane</code> .
<i>metafile</i>	A metafile, as described in <code>with-internal-metafile</code> .
<i>x,y</i>	Integers.
<i>width,height</i>	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 GTK+ or X11/Motif.

Examples There is an example in `examples/capi/graphics/metafile.lisp`.

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

strained, so specifying a *max-width* can also reduce the actual height of the image. The default values of *max-width* and *max-height* are both `nil`.

*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 GTK+ or 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	<p>The <i>display-callback</i> is called with the output pane to draw on, the <code>drawn-pinboard-object</code> itself, and the <i>x</i>, <i>y</i>, <i>width</i> and <i>height</i> of the object, and it is expected to redraw that section.</p> <p>An alternative way of doing this is to create a subclass of <code>pinboard-object</code> and to provide a method for <code>draw-pinboard-object</code>.</p>
Example	<pre>(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)))</pre>
See also	<code>pinboard-layout</code>

**draw-pinboard-object***Generic Function*

Summary	Draws a pinboard object.
Package	<code>capi</code>
Signature	<code>draw-pinboard-object</code> <i>pinboard object</i> &key <i>x y width height</i> &allow-other-keys
Description	<p>The generic function <code>draw-pinboard-object</code> is called whenever a pinboard object needs to be drawn. The <i>x</i>, <i>y</i>, <i>width</i> and <i>height</i> arguments indicate the region that needs to be redrawn, but a method is free to ignore these and draw the complete object.</p>

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 Draws highlighting on a pre-drawn pinboard object.

Package `capi`

Signature `draw-pinboard-object-highlighted pinboard object &key  
&allow-other-keys`

Description The generic function `draw-pinboard-object-highlighted` 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  
`examples/capi/graphics/circled-graph-nodes.lisp`

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

## **draw-pinboard-object-unhighlighted**

*Generic Function*

Summary Removes the highlighting from a pinboard object.

Package `capi`

Signature `draw-pinboard-object-unhighlighted pinboard object &key  
&allow-other-keys`

Description	The generic function <code>draw-pinboard-object-unhighlighted</code> 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	<code>draw-pinboard-object-highlighted</code> <code>highlight-pinboard-object</code>

## drop-object-allows-drop-effect-p

*Function*

Summary	Queries whether a dropping operation can be performed with a given effect.
Package	<code>capi</code>
Signature	<code>drop-object-allows-drop-effect-p</code> <i>drop-object effect =&gt; result</i>
Arguments	<i>drop-object</i> A <i>drop-object</i> , as passed to the <i>drop-callback</i> . <i>effect</i> An effect keyword
Values	<i>result</i> A boolean
Description	The function <code>drop-object-allows-drop-effect-p</code> returns non- <code>nil</code> if the dropping operation can be performed with the given effect <i>effect</i> . It returns <code>nil</code> if the dropping operation cannot be performed. See <code>drop-object-drop-effect</code> for information on drop effect keywords.  <b>Note:</b> <code>drop-object-allows-drop-effect-p</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.
See also	<code>drop-object-drop-effect</code> <code>simple-pane</code>

## drop-object-collection-index

*Function*

Summary	Gets the index and relative place in the <code>collection</code> that an object is being dropped over.	
Signature	<code>drop-object-collection-index</code> <i>drop-object</i> => <i>index</i> , <i>placement</i> <code>(setf (drop-object-collection-index <i>drop-object</i>) (values <i>new-index</i> <i>new-placement</i>))</code>	
Arguments	<i>drop-object</i>	A <i>drop-object</i> , as passed to the <i>drop-callback</i> .
	<i>new-index</i>	An integer.
	<i>new-placement</i>	One of <code>:above</code> , <code>:item</code> or <code>:below</code> .
Values	<i>index</i>	An integer.
	<i>placement</i>	One of <code>:above</code> , <code>:item</code> or <code>:below</code> .
Description	<p>The function <code>drop-object-collection-index</code> returns the index and place relative to that index within the <code>collection</code> that the object <i>drop-object</i> is being dropped over. This information is only meaningful when the pane is an instance of <code>list-panel</code> or <code>tree-view</code>.</p> <p>The returned value <i>index</i> is the position in the <code>collection</code> (see <code>get-collection-item</code> or <code>choice-selection</code>). The returned value <i>placement</i> indicates whether the user is dropping above, on or below the item at <i>index</i>.</p> <p>There is also a setf expander that can be called with these two values within the <code>:drag</code> stage of the operation, to adjust where the user will be allowed to drop the object.</p>	
Notes	<code>drop-object-collection-index</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.	

Example	For an example illustrating the use of drag and drop in a choice, see <code>examples/capi/choices/drag-and-drop.lisp</code>
See also	<code>drop-object-collection-item</code>

**drop-object-collection-item***Function*

Summary	Gets the item and relative place in the <code>collection</code> that an object is being dropped over.	
Signature	<code>drop-object-collection-item <i>drop-object</i> =&gt; <i>item</i>, <i>placement</i></code> <code>(setf (drop-object-collection-item <i>drop-object</i>) (values <i>new-item</i> <i>new-placement</i>))</code>	
Arguments	<i>drop-object</i>	A <i>drop-object</i> , as passed to the <i>drop-callback</i> .
	<i>new-item</i>	An item of a <code>collection</code> .
	<i>new-placement</i>	One of <code>:above</code> , <code>:item</code> or <code>:below</code> .
Values	<i>item</i>	An item of a <code>collection</code> .
	<i>placement</i>	One of <code>:above</code> , <code>:item</code> or <code>:below</code> .
Description	<p>The function <code>drop-object-collection-item</code> returns the item and place relative to that item within the <code>collection</code> that the object <i>drop-object</i> is being dropped over. This information is only meaningful when the pane is an instance of <code>list-panel</code> or <code>tree-view</code>.</p> <p>The returned value <i>placement</i> indicates whether the user is dropping above, on or below the item.</p> <p>There is also a <code>setf</code> expander that can be called with these two values within the <code>:drag</code> stage of the operation, to adjust where the user will be allowed to drop the object.</p>	

Notes	<code>drop-object-collection-item</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.
Example	For an example illustrating the use of drag and drop in a choice, see <code>examples/capi/choices/drag-and-drop.lisp</code>
See also	<code>drop-object-collection-index</code>

## 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	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:	
	<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.

`:none` No dragging is possible.

**Note:** `drop-object-drop-effect` should only be called within a *drop-callback*. It 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`

## drop-object-get-object

*Function*

Summary Returns a dropped object in a given format

Package `capi`

Signature `drop-object-get-object` *drop-object* *format* => *object*

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

Values *object* An object in the given format

Description The function `drop-object-get-object` returns the dropped object in the given format. See `set-drop-object-supported-formats` for information on format keywords.

**Note:** `drop-object-get-object` should only be called within a *drop-callback*. It 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 `set-drop-object-supported-formats`  
`simple-pane`

## drop-object-pane-x drop-object-pane-y

*Functions*

Summary	Gets the coordinates in the pane that an object is being dropped over.
Package	<code>capi</code>
Signature	<code>drop-object-pane-x <i>drop-object</i> =&gt; <i>x-coord</i></code> <code>drop-object-pane-y <i>drop-object</i> =&gt; <i>y-coord</i></code>
Arguments	<i>drop-object</i> A <i>drop-object</i> , as passed to the <i>drop-callback</i> .
Values	<i>x-coord</i> , <i>y-coord</i> Integers.
Description	The functions <code>drop-object-pane-x</code> and <code>drop-object-pane-y</code> return the x and y coordinates within the pane that the object is being dropped over. This information is only meaningful when the pane is an instance of <code>output-pane</code> or one of its subclasses.
Notes	<code>drop-object-pane-x</code> and <code>drop-object-pane-y</code> should only be called within a <i>drop-callback</i> . They are not supported on X11/Motif. See <code>simple-pane</code> for information about drop callbacks.
See also	<code>simple-pane</code>

## 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 <i>drop-object format</i> =&gt; <i>result</i></code>

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><b>Note:</b> <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	<p>The drawing style of the cursor in the Echo Area of an inactive window in the LispWorks IDE.</p> <p>The allowed values are <code>:inverse</code>, <code>:outline</code>, <code>:underline</code> and <code>:invisible</code>.</p>	

## echo-area-pane

*Class*

Summary	The class of the Editor's echo area.
Package	<code>capi</code>
Superclasses	<code>editor-pane</code>
Description	<p>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 LispWorks IDE development environment.</p> <p>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>.</p>

## \*editor-cursor-color\*

*Variable*

Summary	The background color of the cursor.
Package	<code>capi</code>
Initial Value	<code>nil</code>
Description	<p>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 Color System" in the <i>LispWorks CAPI User Guide</i> for information about colors in LispWorks.</p> <p>The value <code>nil</code> means that the cursor background color is the same as the foreground color of the editor pane.</p>
Example	<pre>(setf capi:*editor-cursor-color* :red)</pre>

**\*editor-cursor-active-style\*** *Variable*

Summary	The drawing style of the editor's cursor when the window is active.
Package	<code>capi</code>
Initial Value	<code>:inverse</code>
Description	The drawing style of an <code>editor-pane</code> cursor when the window is active.  The allowed values are <code>:inverse</code> , <code>:outline</code> , <code>:underline</code> , <code>:left-bar</code> and <code>:caret</code> .
See also	<code>editor-pane-blink-rate</code>

**\*editor-cursor-drag-style\*** *Variable*

Summary	The drawing style of the editor's cursor during a selection drag.
Package	<code>capi</code>
Initial Value	<code>:left-bar</code>
Description	The drawing style of an <code>editor-pane</code> cursor during a selection drag.  The allowed values are <code>:inverse</code> , <code>:outline</code> , <code>:underline</code> , <code>:left-bar</code> and <code>:caret</code> .

**\*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 *LispWorks Guide To The Editor*.

**Package** `capi`

**Superclasses** `output-pane`

**Subclasses** `interactive-pane`  
`collector-pane`

**Initargs**

- `:text` A string or `nil`.
- `:enabled` `t`, `nil` OR `:read-only`.
- `:buffer-modes` A list specifying the modes of the editor buffer.
- `:buffer-name` The name of the editor buffer.
- `:change-callback`  
A function designator, or `nil`.
- `:before-input-callback`  
A function designator, or `nil`.
- `:after-input-callback`  
A function designator, or `nil`.

<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

`enabled` controls how user input affects the `editor-pane`. If `enabled` is `nil`, all input from the mouse and keyboard is ignored. When `enabled` is `t`, all input is processed according to the *input-model*. When `enabled` is `:read-only`, input to the pane by keyboard or mouse gestures cannot change the text. More accurately, input via the default *input-model* of `editor-pane` cannot change the text. The **Cut** and **Paste** menu entries are also disabled. When a user tries to change the text, the operation quietly aborts. Programmatic modifications of the text are still allowed (see Notes below for more detail).

The `enabled` state can be set by the accessor `editor-pane-enabled`. `capi:simple-pane-enabled` has the same effect when applied to an `editor-pane`.

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.

A non-empty string value of *text* specifies the initial text displayed. Otherwise an existing editor buffer is displayed. The accessor `editor-pane-text` is provided to read and write the text in the editor 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. *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:

<code>t</code>	Normal wrapping. Display as many characters as possible in the <code>editor-pane</code> line.
<code>nil</code>	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`.

The input behaviour of an `editor-pane` is determined by its *input-model* (inherited from `output-pane`). By default, an `editor-pane` has an *input-model* that implements the functionality of the Editor tool in the LispWorks IDE, and always does it via `call-editor`. You can modify this behavior by changing the *input-model* either by supplying `:input-model` when you call `make-instance`, or by changing the *input-model* later with the accessor `(setf capi:output-pane-input-model)`. It is possible to achieve a minor modification to the default input behavior by prepending the modification (see the example below). Note that functions performing editor operations must do this via `call-editor`.

**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 LispWorks IDE 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*.

## Notes

1. For an `editor-pane` with *enabled* `:read-only`, Editor commands (predefined, and user-defined by `editor:defcommand`) may or may not be able to change the text,

depending on how they are called. When executed by a key sequence they cannot change the text directly. However Editor commands can also be called via `editor:process-character` or `call-editor`, and then are programmatic input and so can change the text.

2. The effect of `enabled:read-only` is on the `editor-pane`. It does not affect the underlying Editor buffer, which can still be modified from other panes. The buffer that is displayed can be changed, and this does not affect the enabled state of the `editor-pane`.

Compatibility  
note

In LispWorks 4.4 and previous versions `editor-pane` supports only fixed-width fonts.

On Cocoa, `editor-pane` supports only fixed-width fonts.

In LispWorks 6.0 and later, variable-width fonts can also be used on Microsoft Windows, GTK+ and Motif. Specify the font via the `:font` initarg (see `simple-pane`).

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))

```

This example makes an editor-pane with no input behavior:

```

(capi:contain
 (make-instance 'capi:editor-pane :input-model nil))

```

This example makes an editor-pane with the default input behavior, except that pressing the mouse button displays a message rather than setting the point:

```
(progn
  (defun foo (self x y)
    (capi:display-message "Button-1 Press at ~a/~a"
                          x y))
  (let ((ep (make-instance 'capi:editor-pane)))
    (setf (capi:output-pane-input-model ep)
          (list* '(:button-1 :press) foo)
          (capi:output-pane-input-model ep)))
    (capi:contain ep)))
```

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	capi	
Signature	editor-pane-blink-rate <i>self</i> => <i>blink-rate</i>	
Arguments	<i>self</i>	An editor pane.
Values	<i>blink-rate</i>	A non-negative real number, or nil.
Description	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.	

If *blink-rate* is a positive real number, then it is the blinking rate in milliseconds. If *blink-rate* is 0, then there is no blinking. If *blink-rate* is `nil`, then the default blinking rate is used.

The default method on `editor-pane-blink-rate` returns `nil`, which means use the default blinking rate. `set-default-editor-pane-blink-rate`.

You can define your own methods on `editor-pane-blink-rate` for `editor-pane` and subclasses thereof.

See also `*editor-cursor-active-style*`  
`editor-pane`  
`editor-pane-native-blink-rate`  
`set-default-editor-pane-blink-rate`

## 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	<code>editor-pane-blink-rate</code> <code>set-default-editor-pane-blink-rate</code>	

**editor-pane-selected-text***Function*

Summary	Returns the selected text in an <code>editor-pane</code> .	
Package	<code>capi</code>	
Signature	<code>editor-pane-selected-text</code> <i>editor-pane</i> => <i>result</i>	
Arguments	<i>editor-pane</i>	An <code>editor-pane</code> .
Values	<i>result</i>	A string or <code>nil</code> .

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**

*Generic 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 generic 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	<i>editor-pane</i>	An <code>editor-pane</code> .
Values	<i>stream</i>	An output stream.
Description	The function <code>editor-pane-stream</code> returns the stream where the results of evaluation in the editor buffer currently associated with <i>pane</i> are printed to.	
See also	<code>editor-pane</code>	

## **editor-window**

*Generic Function*

Summary	Returns the editor window object.	
Package	<code>capi</code>	
Signature	<code>editor-window editor =&gt; editor-window</code>	
Arguments	<i>editor</i>	An <code>editor-pane</code> or an Editor interface in the LispWorks IDE.
Values	<i>editor-window</i>	An editor window object.
Description	The generic function <code>editor-window</code> returns the editor window object associated with <i>editor</i> .  The functionality of editor windows is documented in the <i>LispWorks Editor User Guide</i> .	
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> .
	<code>:widget-name</code>	A string designator.

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.

`:visible-min-width`  
The minimum visible width of the element.

`:visible-min-height`  
The minimum visible height of the element.

`:visible-max-width`  
The maximum visible width of the element.

`:visible-max-height`  
The maximum height of the element.

`: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

`element-parent`  
`element-widget-name`

Readers

`element-interface`  
`help-key`

Description

The class `element` contains the slots `parent` and `interface` which contain the element and the interface that the element is contained in respectively. The writer method `element-parent` can be used to re-parent an element into another parent (or to remove it from a container entirely by setting its parent to `nil`). Note that an element should not be used in more than one place at a time.

The initarg *accepts-focus-p* specifies that the element can accept input. The default value is `t`. In some subclasses including `display-pane` and `title-pane` the default value of *accepts-focus-p* is `nil`. A pane accepts the input focus if and only if the function `accepts-focus-p` returns true.

*accepts-focus-p* also influences whether a pane is a tabstop on Microsoft Windows, where a pane acts as a tabstop if and only if the function `accepts-focus-p` returns true and the `: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.

*help-key* is used to determine how help is displayed for the pane. The value `nil` means that no help is displayed. Otherwise, *help-key* is passed to the *help-callback*, except when *help-key* is `t`, when the name of the pane is passed to the *help-callback*. For details of *help-callback*, see `interface`.

*widget-name* specifies the widget name of the element. This is used to match resources on GTK+ and Motif. Note that this name will be in the path only if the element has a representation. `tab-layout` and `pinboard-layout` always have a representation, as do all elements that show anything on the screen. Other layouts may or may not have a representation and so you should not supply *widget-name* for these.

The actual widget name is the result of a call to `c1:string`, except when *widget-name* is a symbol, in which case the symbol name is downcased to derive the widget name.

If *widget-name* is not supplied, the system constructs a default widget name which is the name of the class of the widget (downcased), except for top level interfaces on GTK+ where the *application-class* is prepended followed by a dot.

Example GTK+ resource files are in `lib/6-0-0-0/examples/gtk/`

**Note:** When *widget-name* is supplied, the GTK+ library does not prepend the *application-class*.

The accessor `element-widget-name` gets and (with `setf`) sets the *widget-name*. *widget-name* is used when the widget is created, that is when `display` is called on the top level interface of the element. Setting *widget-name* afterwards has no effect.

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 `nil` (meaning no maximum), but the hints can be specified to change these values. The possible values for these hints are as follows:

<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.

### Example

```
(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/Setup.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 <i>element</i> =&gt; <i>container</i></code>
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 screen 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	<code>document-frame</code> <code>element</code>

## 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 <i>element</i> =&gt; <i>interface</i></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 `attach-interface-for-callback`  
`element`

## **element-screen**

*Function*

Summary Returns the screen that an element is associated with.

Package `capi`

Signature `element-screen element => screen`

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	<code>capi</code>	
Signature	<code>ensure-area-visible self x y width height</code>	
Arguments	<code>self</code>	A <code>simple-pane</code> with internal scrolling.
	<code>x,y</code>	The coordinates of the origin of the area to make visible.
	<code>width, height</code>	The dimensions of the area to make visible
Description	The generic function <code>ensure-area-visible</code> ensures that the area specified by <code>x</code> , <code>y</code> , <code>width</code> and <code>height</code> , or at least part of it, is visible.	
	This function works only for subclasses of <code>simple-pane</code> that do internal scrolling (such as <code>editor-pane</code> ). An error is signalled if it is called with other classes.	

## ensure-interface-screen

*Function*

Summary	The <code>ensure-interface-screen</code> function ensures that a top level interface is displayed on a given screen.	
Package	<code>capi</code>	
Signature	<code>ensure-interface-screen self &amp;key screen</code>	
Description	This ensures that the top level interface is displayed on the given <code>screen</code> (or the default) if <code>display</code> is called later without a <code>screen</code> 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	<code>capi</code>	
Signature	<code>execute-with-interface <i>interface</i> <i>function</i> &amp;rest <i>args</i></code>	
Arguments	<i>interface</i>	An interface
	<i>function</i>	A function designator
	<i>args</i>	Arguments passed to <i>function</i>
Description	The function <code>execute-with-interface</code> is a useful way of operating on an <code>interface</code> 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).	
Notes	<ol style="list-style-type: none"> <li><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>.</li> <li>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.</li> </ol>	

3. `execute-with-interface` calls *function* on the current process if *interface* does not have a process.

Example

```
(setq a (capi:display (make-instance 'capi:interface)))  
  
(capi:execute-with-interface  
  a 'break  
  "Break inside the interface process")
```

See also

```
apply-in-pane-process  
execute-with-interface-if-alive
```

## execute-with-interface-if-alive

*Function*

Summary Executes a function in the event process of a given interface if it is alive.

Package `capi`

Signature `execute-with-interface-if-alive interface function &rest args => nil`

Description The function `execute-with-interface-if-alive` applies the function *function* to the arguments *args* in the process of the interface *interface*, if the interface is "alive". An interface is alive if it has a representation on the screen.

If *interface* is not alive, *function* is not applied. This is in contrast to `execute-with-interface`, which in this case applies the function in the current process.

`execute-with-interface-if-alive` 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.

**Note:** All accesses (reads as well as writes) on a CAPI interface and its sub-elements should be performed in the interface process.

See also `execute-with-interface`

## exit-confirmer

*Function*

**Summary** The `exit-confirmer` function is called by the **OK** button on a dialog created with `popup-confirmer`.

**Package** `capi`

**Signature** `exit-confirmer &rest dummy-args`

**Description** This is the function that is called by the **OK** button on a dialog created using `popup-confirmer`, 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 `popup-confirmer`.

**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 hierarchical list of items which (unlike `tree-view`) allows extended selection.

Package            `capi`

Superclasses      `tree-view`

Description        The class `extended-selection-tree-view` is like `tree-view` but allows more than one item to be selected at once.

Notes              Although `extended-selection-tree-view` is a subclass of `collection`, it does its own items handling and you must not access its *items* and related slots directly. In particular for `extended-selection-tree-view` do not pass `:items`, `:items-count-function`, `:items-get-function` OR `:items-map-function`, and do not use the corresponding accessors.

See also            `tree-view`

## **filtering-layout**

*Class*

Summary            A layout that can be used for filtering.

Package            `capi`

Superclasses	<code>row-layout</code>
Initargs	<code>:callback-object</code> The argument for the callbacks. If it is <code>nil</code> the top-level-interface of the layout is used.
	<code>:change-callback</code> 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.
	<code>:callback</code> A function of one argument (the <i>callback-object</i> ). It is called when the user presses <b>Return</b> , makes a selection from the menu, or clicks the <b>Confirm</b> button. If <i>callback</i> is not supplied, <i>change-callback</i> is called instead.
	<code>:text</code> A string specifying the initial text of the filter, or <code>nil</code> .
	<code>:matches-title</code> A string, <code>t</code> or <code>nil</code> .
	<code>:help-string</code> A string, <code>t</code> or <code>nil</code> .
	<code>:label-style</code> <code>:short</code> , <code>:medium</code> or <code>:long</code> .
Accessors	<code>filtering-layout-state</code> <code>filtering-layout-matches-text</code>
Description	<p>The main part of a filtering layout is a <code>text-input-pane</code> 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:</p> <ul style="list-style-type: none"> <li>• the string is used as a regular expression or plain string</li> <li>• the filter excludes matches or includes matches</li> </ul>

- 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.

`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. If `matches-title` is `t` 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)`.

If *help-string* is non-nil then the filter has a Help button which raises a default help text if *help-string* is  $\epsilon$ , or the text of *help-string* if it is a string.

If *label-style* is `:short` the filter menu has a short title. For example if the filter is set for case-sensitive plain inclusive matching the short label is **PMC**. If *label-style* is `:medium` then this label would be **Filter:C**. Any other value of *label-style* would make a long label **Plain Match Cased** .

## 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 =&gt; 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	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.  <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> .  See <code>filtering-layout</code> for details.
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 =&gt; 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		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> .  If there is no such edge, <code>find-graph-edge</code> returns <code>nil</code> .
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>cap</code>	
Signature		<code>find-graph-node graph object =&gt; 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		The generic function <code>find-graph-node</code> finds the node that corresponds to the item <i>object</i> .  If there is no such node, <code>find-graph-node</code> returns <code>nil</code> .
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"><li>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.</li><li>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.</li></ol> <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 LispWorks IDE 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	<p><code>:vertical-gap</code> The gap between rows in the form.</p> <p><code>:vertical-adjust</code> The adjustment made to the rows.</p> <p><code>:title-gap</code> The gap between the two columns.</p> <p><code>:title-adjust</code> The adjustment made to the left column.</p>
Accessors	<p><code>form-vertical-gap</code></p> <p><code>form-vertical-adjust</code></p> <p><code>form-title-gap</code></p> <p><code>form-title-adjust</code></p>
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.
Example	<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-metafile

*Function*

Summary Frees a metafile.

Package `capi`

Signature `free-metafile metafile`

Arguments `metafile` A metafile.

Description The function `free-metafile` releases the window system storage used by the metafile.

`free-metafile` must be called when the metafile is no longer needed, to avoid memory leaks.

`free-metafile` is not implemented on GTK+ or X11/Motif.

Examples There is an example in `examples/capi/graphics/meta-file.lisp`.

See also `clipboard`  
`draw-metafile`  
`draw-metafile-to-image`

## free-sound

*Function*

Summary Frees a loaded sound object.

Package	<code>capi</code>
Signature	<code>free-sound <i>sound</i></code>
Arguments	<code><i>sound</i></code> An array returned by <code>load-sound</code> .
Description	The function <code>free-sound</code> unloads (frees) the loaded sound object <code><i>sound</i></code> .
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 <i>self index</i></code>
Description	The generic function <code>get-collection-item</code> returns the item at position <code><i>index</i></code> from the <code>collection <i>self</i></code> . It achieves this by calling the <code><i>items-get-function</i></code> 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 <i>element</i></code>
Description	<p>The function <code>get-constraints</code> returns the constraints for <i>element</i> as multiple values (the values are the minimum width, the minimum height, the maximum width and the maximum height).</p> <p>This function calls the generic function <code>calculate-constraints</code> to calculate these sizes initially, but then just uses the values in the geometry cache for the element. To force an element to take account of its new constraints, call the function <code>invalidate-pane-constraints</code>.</p>
See also	<p><code>calculate-constraints</code>  <code>define-layout</code>  <code>element</code>  <code>invalidate-pane-constraints</code></p>

## get-horizontal-scroll-parameters

*Generic Function*

Summary	Queries the scroll parameters of a horizontal scroll bar.	
Package	<code>capi</code>	
Signature	<code>get-horizontal-scroll-parameters <i>self</i> &amp;rest <i>keys</i> =&gt; <i>parameter, parameter,...</i></code>	
Arguments	<i>self</i>	A displayed <code>simple-pane</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**      Retrieves the specified parameters of the horizontal scroll bar of *self*, which should be a displayed instance of a subclass of `simple-pane` which does internal scrolling (such as `editor-pane`).

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-panel`, 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	capi	
Signature	<code>get-vertical-scroll-parameters self &amp;rest keys =&gt; 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"><li><code>:min-range</code>     The minimum data coordinate.</li><li><code>:max-range</code>     The maximum data coordinate.</li><li><code>:slug-position</code>                   The current scroll position.</li><li><code>:slug-size</code>      The length of the scroll bar slug.</li><li><code>:page-size</code>      The scroll page size.</li><li><code>:step-size</code>      The scroll step size.</li></ul> <p><b>Note:</b> For the other pane classes, such as <code>list-panel</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 `graph-node-x`  
`graph-node-y`  
`graph-node-width`  
`graph-node-height`  
`graph-node-in-edges`  
`graph-node-out-edges`

Description The default class of nodes in a `graph-pane`.  
The `graph-pane` generates a graph of `graph-node` and `graph-edge` objects.

See also `graph-edge`  
`graph-pane`

## **graph-node-children**

*Generic Function*

Summary Returns the children of a graph node.

Package `capi`

Signature `graph-node-children node => result`

Arguments *node* A `graph-node`.

Values *result* A list.

Description The generic function `graph-node-children` returns a list of all the 'children' of the node *node*. These children are the nodes which are at the other end of some edge in the `graph-node-out-edges` of the `graph-node` *node*.

See also `graph-node`

## **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	<p><code>:roots</code>           The roots of the graph.</p> <p><code>:children-function</code>                       Returns the children of a node.</p> <p><code>:layout-function</code>                       A keyword denoting how to layout the nodes.</p>

`: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.

Example

```
(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	<code>capi</code>	
Signature	<code>graph-pane-add-graph-node</code> <i>graph-pane object parent-node</i> => <i>new-node</i>	
Arguments	<i>graph-pane</i>	A <code>graph-pane</code> .
	<i>object</i>	An object.
	<i>parent-node</i>	A <code>graph-node</code> .
Values	<i>new-node</i>	A <code>graph-node</code> .
Description	The generic function <code>graph-pane-add-graph-node</code> adds a new node in the graph <i>graph-pane</i> corresponding to <i>object</i> , and links it as a child of <i>parent-node</i> .	
See also	<code>graph-node</code> <code>graph-pane</code>	

## graph-pane-delete-object

*Generic Function*

Summary	Removes a node from a graph.	
Package	capi	
Signature	<code>graph-pane-delete-object</code> <i>graph-pane object</i>	
Arguments	<i>graph-pane</i>	A <code>graph-pane</code> .
	<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	capi	
Signature	<code>graph-pane-delete-objects</code> <i>graph-pane objects</i>	
Arguments	<i>graph-pane</i>	A <code>graph-pane</code> .
	<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*  
(`setf graph-pane-direction`) *direction* *graph-pane* => *direction*

Arguments *graph-pane* A `graph-pane`.

Values *direction* One of `:forwards` or `:backwards`.

Description     The generic function `graph-pane-direction` returns the direction of the graph *graph-pane*. If the *layout-function* of *graph-pane* is `:top-down` OR `:left-right` then *direction* is `:forwards`. Otherwise *direction* is `:backwards`.

                  The generic function `(setf graph-pane-direction)` maintains the dimension of the *layout-function* but potentially reverses its direction.

Example

```
(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
```

See also        `graph-pane`

## graph-pane-edges

*Function*

Summary        Returns the edges of a graph.

Package        `capi`

Signature      `graph-pane-edges graph-pane => edges`

Arguments      *graph-pane*        A `graph-pane`.

Values         *edges*             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.

`:rows`           The number of rows in the grid.

`:x-ratios`        The ratios between the columns.

`:y-ratios`        The ratios between the rows.

`:x-gap`           The gap between each column.

`:y-gap`           The gap between each row.

`:x-uniform-size-p`  
                   If `t`, make each of the columns the same size.

`:y-uniform-size-p`  
                   If `t`, 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.

Normally, the items in a `grid-layout` are arranged to look like a set of columns that are joined horizontally and rows that are joined vertically. All the cells in each column have the same width and all the cells in each row have the same height. The keyword `:right-extend` (or `:bottom-extend`) can be used to allow an item to span more than one column (or row). The keyword should be placed in the cell of the *description* that you want the item to expand into. For `:right-extend`, the cell immediately to the left will be

extended to fill both columns in that row. For `:bottom-extend`, the cell immediately above will be extended to fill both rows in that column.

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")

```

This example illustrates the use of `:right-extend` and `:bottom-extend` to make cells span multiple columns and rows:

`examples/capi/layouts/extend.lisp`

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 `hide-interface` hides the interface containing a specified pane.

**Package** `capi`

**Signature** `hide-interface pane &optional iconify`

**Description** The function `hide-interface` hides the interface containing *pane* from the screen. If *iconify* is non-nil then it will iconify it, else it will just remove it from the screen. To show it again, use `show-interface`.

The default value of *iconify* is `t`.

See also `interface`  
`show-interface`  
`quit-interface`

## hide-pane

*Generic Function*

**Summary** Hides the specified pane.

Package	<code>capi</code>
Signature	<code>hide-pane <i>pane</i> =&gt; <i>pane</i></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 <i>pinboard object</i> &amp;key <i>redisplay</i></code>
Arguments	<i>pinboard</i> A <code>pinboard-layout</code> . <i>object</i> A <code>pinboard-object</code> . <i>redisplay</i> A generalised boolean.
Description	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.  The pinboard object highlighting is drawn according to the <i>highlight-style</i> of the <code>pinboard-layout</code> <i>pinboard</i> .  If <i>redisplay</i> is non-nil the highlighting is drawn immediately. The default value for <i>redisplay</i> is <code>t</code> .

See also `unhighlight-pinboard-object`  
`draw-pinboard-object-highlighted`  
`pinboard-object`  
`pinboard-layout`

## image-list

*Class*

Summary An object used to manage the images displayed by tree views and list views.

Package `capi`

Superclasses `capi-object`

Initargs  
`:image-width` The width of the images in this image list.  
`:image-height` The height of the images in this image list.  
`:image-sets` A list of images or image sets.

Description The `:image-sets` initarg specifies a list. Each item in the list *image-sets* may be one of the following.

A pathname or string

This specifies the filename of a file suitable for loading with `load-image`.

A symbol The symbol must be a predefined image identifier, or have been registered by means of a call to `register-image-translation`.

An image object, as returned by `load-image`.

An image-set object

See `image-set` for further details.

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.

The images added to the image list are numbered in order, starting from zero. An image-set containing  $n$  images contributes  $n$  images to the image list, and hence consumes  $n$  consecutive integer indices.

Example	See the files <code>examples/capi/choice/tree-view.lisp</code> <code>examples/capi/choice/extended-selection-tree-view.lisp</code>
See also	<code>image-set</code> <code>load-image</code> <code>register-image-translation</code>

## 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/Setup.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/Setup.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 GTK+ and Motif.

*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 `capi`

Signature `installed-libraries => libraries`

Values *libraries* A list of library names.

Description The function `installed-libraries` returns the list of installed CAPI libraries.

A library name is a keyword naming a library.

On Linux, FreeBSD and x86/x64 Solaris platforms, `libraries` is initially `(:gtk)` but may also include `:motif` if the deprecated "capi-motif" module is loaded.

On Microsoft Windows platforms, currently *libraries* is always `(:win32)`.

On Mac OS X platforms, in the native GUI image *libraries* is always `(:cocoa)`. In the Mac OS X/GTK+ image, `libraries` is initially `(:gtk)` but may also include `:motif` if the deprecated "capi-motif" module is loaded.

In LispWorks for UNIX only (not LispWorks for Linux, FreeBSD, or x86/x64 Solaris), currently *libraries* is always `(:motif)`.

See also `default-library`

## interactive-pane

*Class*

**Summary** An `interactive-pane` is an editor with a process reading and processing input, and that collects any output into itself. The class `listener-pane` is built upon this, and adds functionality for handling Lisp forms.

**Package** `capi`

**Superclasses** `editor-pane`

**Subclasses** `listener-pane`  
`shell-pane`

**Initargs** `:top-level-function`

The input processing function.

**Readers** `interactive-pane-stream`  
`interactive-pane-top-level-function`

**Description** An `interactive-pane` contains its own GUI stream. The *top-level-function* is called once, when the interactive pane is created: it needs to repeatedly take input from the GUI stream and write output to it.

The first argument to *top-level-function* 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 *top-level-function* is a function which runs a Lisp listener top-loop.

**Compatibility note** This class was named `interactive-stream` 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 `interactive-stream` and its accessors `interactive-stream-top-level-function` and `interactive-stream-stream` 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 <code>interactive-pane</code> .
Package	<code>capi</code>
Signature	<code>interactive-pane-execute-command</code> <i>interactive-pane</i> <i>command</i> &key <i>command-modification-function</i> <i>editp</i> &allow-other-keys
Arguments	<i>interactive-pane</i> An <code>interactive-pane</code> . <i>command</i> A Lisp form. <i>command-modification-function</i> A function or <code>nil</code> . <i>editp</i> A generalized boolean.
Description	<p>The generic function <code>interactive-pane-execute-command</code> has the same effect as the user typing the Lisp form <i>command</i> into the <code>interactive-pane</code> <i>interactive-pane</i>, and pressing <b>Return</b>.</p> <p><code>interactive-pane-execute-command</code> may be called from any process.</p> <p>If <i>command-modification-function</i> is non-<code>nil</code>, it is a function of one argument. It is called with argument <i>command</i> in the process in which <i>interactive-pane</i> runs. The result of this call is used as the command to enter. The default value of <i>command-modification-function</i> is <code>nil</code>.</p> <p>If <i>editp</i> is true then the command is left at the end of the pane for the user to edit before pressing <b>Return</b>. If <i>editp</i> is <code>nil</code> then <code>interactive-pane-execute-command</code> simulates the user pressing <b>Return</b>. The default value of <i>editp</i> is <code>nil</code>.</p>
See also	<code>interactive-pane</code> <code>listener-pane-insert-value</code>

**interface***Class*

Summary	The class <code>interface</code> 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	<code>capi</code>	
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, before display and user interaction.
	<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 <i>x</i> position for the interface.
	<code>:best-y</code>	The best <i>y</i> position for the interface.
	<code>:best-width</code>	The best width of the interface.
	<code>:best-height</code>	The best height of the interface.

- :geometry-change-callback**  
A function called when the interface geometry changes.
- :activate-callback**  
A function called when the interface is activated or deactivated.
- :iconify-callback**  
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.  
*override-cursor* is not supported on Cocoa.  
*override-cursor* is ignored by *text-input-pane* on GTK+.
- :message-area** A boolean determining whether the interface has a message area.
- :enable-pointer-documentation**  
A boolean determining whether Pointer Documentation is enabled.  
*enable-pointer-documentation* is supported only on Motif. It is possible to implement equivalent functionality for *output-pane* and subclasses such as *pinboard-layout* by using the *focus-callback* of *output-pane*.
- :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, later versions of Microsoft Windows, and GTK+.
- `:window-styles`  
A list of keywords, or `nil`.
- `:toolbar-items`  
A list of items for the toolbar.
- `:toolbar-states`  
A toolbar state plist.
- `:default-toolbar-states`  
A toolbar state plist.

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 interface-toolbar-items interface-toolbar-states interface-default-toolbar-states </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 <b>Works</b> menu in the Lisp-Works IDE). To switch these automatic menus off, pass <code>:auto-menus nil</code>.</p>

When you have an instance of an interface, you can display it either as an ordinary window or as a dialog using respectively `display` and `display-dialog`. The CAPI calls *create-callback* (if supplied) with the interface as its single argument, 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. *override-cursor* is not supported on Cocoa. *override-cursor* is ignored by `text-input-pane` on GTK+.

If *message-area* is true, then the interface is created with 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, on Motif. 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:

- `:tooltip` A tooltip is requested. The function needs to return a string to display in the tooltip, or `nil` if no tooltip should be displayed.
- `:help` The function should display a detailed, asynchronous help. This value is passed when the user presses the `F1` key (not implemented on Cocoa). `:help` is also passed when the user clicks the '?' box in the title bar of a Microsoft Windows dialog with window style `:contexthelp` (see *window-styles* below).

On Motif only, *type* can also be one of:

- `:pointer-documentation-enter`  
The cursor entered the pane. The function should set the pointer documentation.
- `:pointer-documentation-leave`  
The cursor left the pane. The function needs to reset the pointer documentation.

*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 use 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. It also works on GTK+, provided that GTK+ and the X server support it. On GTK+ it is supported in version 2.12 and later. The X server needs compositing manager to do it. *: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 and GTK+: The window is only visible when it is the active window.

`:toolbox`

Cocoa, Microsoft Windows and GTK+: A window with a small title bar. This window style is used in `docking-layout`.

`:borderless`

Cocoa, Microsoft Windows, GTK+ 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, GTK+ 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, Microsoft Windows and GTK+: The window is always above all other windows. Such a window is also known as a windoid.

`:ignores-keyboard-input`

Cocoa and GTK+: 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 allowing resizing.)

`:contexthelp`

Microsoft Windows: A '?' box appears in the window's title bar that sends *help-callback* type `:help`.

If *toolbar-items* is non-nil, then the interface will have a toolbar, which is typically displayed at the top of the window. The value of *toolbar-items* is a list of objects of type `toolbar-button`, `toolbar-component` or `simple-pane`, which are items that might be shown on the toolbar. The set of visible items, their order and their appearance is determined by the current *toolbar-state*, which can be changed if the user customizes the toolbar interactively. Each `toolbar-button` or `simple-pane` in the *toolbar-items* list (including those within a `toolbar-component`) should have a *name* that is not `eq1` to any other item in the list. Each `toolbar-button` should have *image* and *text* specified, to control the image and title that is shown for the item. Each `simple-pane` should have *toolbar-title* specified, to control the title that is shown for the item.

*toolbar-states* is a plist containing information about the state of the toolbar. The user can also change this by customizing the toolbar, so you cannot assume that the value will be the same each time you read it. See `interface-toolbar-state` for a description of the keys and values in this plist.

*default-toolbar-states* is a plist containing information about the default state of the toolbar, which you can provide as the suggested toolbar state for the interface. The `:items` key will be used in the Customize dialog as the "default" set of toolbar buttons. If both *default-toolbar-states* and *toolbar-states* are supplied, then the value of any key in *toolbar-states* takes precedence over that of the same key in *default-toolbar-states*. See `interface-toolbar-state` for a description of the keys and values in this plist.

## Notes

1. *create-callback* can only be used for actions that are part of the creation of the pane, that is preparing the pane for display. The *create-callback* is called before the pane is actually displayed, and therefore cannot interact with the user.
2. On Microsoft Windows `F1` always calls *help-callback* if it is non-nil.

3. `(setf capi:interface-message-area)` has an effect only before display. After display, this writer has no effect unless the interface is destroyed and re-created.
4. 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) 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 the interrupt gesture in the editor pane. (For example, the interrupt gesture is `Meta+Control+C` on Motif and `Control+Break` on Microsoft Windows). 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))
```

The code in `examples/capi/applications/simple-symbol-browser.lisp` illustrates the use of *toolbar-items*.

See also

- `layout`
- `switchable-layout`
- `menu`
- `display`
- `display-dialog`
- `interface-display`
- `quit-interface`
- `define-interface`
- `activate-pane`
- `titled-object`
- `interface-toolbar-state`
- `interface-customize-toolbar`

## interface-customize-toolbar

*Function*

**Summary**      Displays a window which allows the user to customize its the toolbar.

**Signature**      `interface-customize-toolbar` *interface*

**Arguments**      *interface*      A CAPI interface.

The function `interface-customize-toolbar` displays a window owned by the interface *interface* that allows the user to customize the toolbar of that interface.

*interface* must be displayed at the time `interface-customize-toolbar` is called.

See also

- `interface`
- `toolbar`

## interface-display

*Generic Function*

Summary	The function called to display an interface on screen.
Package	<code>capi</code>
Signature	<code>interface-display</code> <i>interface</i>
Arguments	<i>interface</i> An instance of a subclass of <i>interface</i> .
Description	<p>The generic function <code>interface-display</code> is called by <code>display</code> to display an interface on screen.</p> <p>The primary method for <i>interface</i> actually does the work. You can add <code>:before</code> methods on your own interface classes for code that needs to be executed just before the interface appears, and <code>:after</code> methods for code that needs to be executed just after the interface appears.</p>
Notes	<ol style="list-style-type: none"><li>1. <code>interface-display</code> is called in the process of <i>interface</i>.</li><li>2. <code>interface-display</code> is not called when <i>interface</i> is displayed as a dialog. Another way to run code before it appears on screen is to supply a <i>create-callback</i> for <i>interface</i>.</li></ol>
Example	This example shows how <code>interface-display</code> 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	<code>capi</code>	
Signature	<code>interface-display-title</code> <i>interface</i> => <i>string</i>	
Arguments	<i>interface</i>	A CAPI interface.
Values	<i>string</i>	A string.
Description	The function <code>interface-display-title</code> returns the title to use when displaying the interface <i>interface</i> 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 `capi`

Signature `interface-extend-title interface title => string`

Arguments	<i>interface</i>	A CAPI <i>interface</i> .
	<i>title</i>	A string.
Description	<p>The generic function <code>interface-extend-title</code> is called by the system with an <i>interface</i> and its <i>title</i> before actually displaying the <i>title</i> on the screen. The result must be a string, which is actually displayed. There is no requirement for any relation between the <i>title</i> argument and the result.</p> <p>The return value <i>string</i> is the <i>title</i> 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	<p><code>interface-display-title</code>  <code>set-default-interface-prefix-suffix</code></p>	

## interface-geometry

## Generic Function

Summary	Returns the geometry of an <i>interface</i> .	
Package	<code>capi</code>	
Signature	<code>interface-geometry <i>interface</i> =&gt; <i>geometry</i></code>	
Arguments	<i>interface</i>	An instance of a subclass of <i>interface</i> .
Values	<i>geometry</i>	A list.
Description	<p>The generic function <code>interface-geometry</code> returns a list representing the geometry of <i>interface</i> in pixel values.</p> <p><i>geometry</i> is of the form <i>(x y width height)</i>.</p>	
See also	<code>interface</code>	

## interface-iconified-p

*Function*

Summary	The predicate for whether an interface is iconified.	
Package	<code>capi</code>	
Signature	<code>interface-iconified-p <i>pane</i> =&gt; <i>iconifiedp</i></code>	
Arguments	<code><i>pane</i></code>	A CAPI element.
Values	<code><i>iconifiedp</i></code>	A boolean.
Description	<p>The function <code>interface-iconified-p</code> returns <code>t</code> if the top level interface containing <code><i>pane</i></code> is iconified. This means that the window is visible as an icon, also referred to as minimized.</p> <p>If the top level interface is not iconified, then <code>interface-iconified-p</code> returns <code>nil</code>.</p>	
See also	<code>hide-interface</code> <code>top-level-interface</code> <code>top-level-interface-display-state</code>	

## interface-keys-style

*Generic Function*

Summary	Determines the emulation for an interface.	
Package	<code>capi</code>	
Signature	<code>interface-keys-style <i>interface</i> =&gt; <i>keys-style</i></code>	
Arguments	<code><i>interface</i></code>	An instance of a subclass of <code>interface</code> .
Values	<code><i>keys-style</i></code>	A keyword, <code>:pc</code> , <code>:emacs</code> or <code>:mac</code> .

Description The generic function `interface-keys-style` returns a keyword indicating a keys style, or *emulation*. It is called when *interface* starts running in a new process, and *keys-style* determines how user input is interpreted by output panes (including `editor-pane`) in *interface*.

The editor (that is, instances of `editor-pane` and its subclasses) responds to user input gestures according to one of three basic models.

When *keys-style* is `:emacs`, the editor emulates GNU Emacs. This value is allowed on all platforms.

When *keys-style* is `:pc`, the editor emulates standard Microsoft Windows keys on Windows, and KDE/Gnome keys on GTK+ and Motif. This value is allowed in the Windows, GTK+ and X11/Motif implementations.

When *keys-style* is `:mac`, the editor emulates Mac OS X editor keys. This value is allowed only in the Mac OS X Cocoa implementation.

The most important differences between the styles are in the handling of the `A1t` key on Microsoft Windows, selected text, and accelerators:

`:emacs` `A1t` is interpreted on Microsoft Windows as the Meta key (used to access many Emacs commands).

The `:meta` modifier is used in an `output-pane input-model` gesture specification.

Control characters such as `ctrl+s` are not interpreted as accelerators.

The selection is not deleted on input.

`: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.

## Notes

On Motif the code to implement accelerators and mnemonics clashes with the LispWorks meta key support. Therefore the keyboard must be configured so that none of the keysyms connected to `mod1` (see `xmodmap`) are listed in the variable

`capi-motif-library:*meta-keySYM-search-list*`, which must be also be non-nil. Note also that Motif requires Alt to be on mod1.

See also `editor-pane`

## interface-match-p

*Generic Function*

**Summary** Determines whether an interface is suitable for displaying `initargs`.

**Package** `capi`

**Signature** `interface-match-p interface &rest initargs &key &allow-other-keys => matchp`

**Arguments** *interface* An instance of a subclass of `interface`.  
*initargs* Initargs for *interface*.

**Values** *matchp* A boolean

**Description** The generic function `interface-match-p` returns a true value if *interface* is suitable for displaying the *initargs*.

`interface-match-p` is used by `locate-interface`. When there is an existing interface for which `interface-match-p` returns true, then `locate-interface` returns it.

The default method for `interface-match-p` always returns `nil`. You can add methods for your own interface classes.

See also `locate-interface`

## 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> =&gt; <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 <code>interface</code>, simply returns <code>(nil nil nil)</code>.</p> <p><b>Note:</b> 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-preserve-state***Generic Function*

Summary	Called before an interface is destroyed during session saving.
Signature	<code>interface-preserve-state <i>interface</i></code>
Arguments	<i>interface</i> An interface.
Description	<p>The generic function <code>interface-preserve-state</code> is called by <code>hcl:save-current-session</code> just before it destroys an interface, on the interface process. You can specialize this for your own interface classes. Your methods should not interact with the user or other external sources, and should not interact with other processes, because it is called after <code>hcl:save-current-session</code> already started to destroy interfaces.</p> <p>The return value is not used.</p> <p>The default method does nothing.</p>
See also	<code>interface-preserving-state-p</code>

**interface-preserving-state-p***Function*

Summary	The predicate for whether an interface is in "preserving-state" context.
Signature	<code>interface-preserving-state-p <i>interface</i> =&gt; <i>result</i></code>
Arguments	<i>interface</i> An interface.
Values	<i>result</i> <code>t</code> , <code>:different-invocation</code> OR <code>:keeping-processes</code> .
Description	An interface enters "preserving-state" context just before it is destroyed by <code>hcl:save-current-session</code> , and exits the context just after <code>interface-display</code> returns.

If the interface *interface* is in "preserving-state" context, the result of `:interface-preserving-state-p` is either `t` or `:different-invocation`. The value `t` means that the current invocation of LispWorks is still the same invocation. The value `:different-invocation` means it is a different invocation, in other words it is the saved image that is restarted.

In other circumstances `interface-preserving-state-p` can return `:keeping-processes`, which means that the interfaces are destroyed but processes that are not associated with *interface* are not killed. That currently happens only on Microsoft Windows when the programmer changes the arrangement of IDE windows via **Preferences... > Environment > General > Window Options**.

`interface-preserving-state-p` is typically used in the *destroy-callback* of an interface or a pane to decide whether really to destroy the information, and in the *create-callback* or `interface-display` to decide whether the existing information can be used. Note that if it is a pane, it needs to find the `top-level-interface`.

Information that is made entirely of Lisp objects can be preserved in all cases. Information that is associated with external objects is invalid when the image is restarted. So when `interface-preserving-state-p` is used inside the *create-callback* or `interface-display`, external information can be preserved only if it returns `t`. When `interface-preserving-state-p` returns `t`, the external information may be preserved, unless it is tied to the lightweight process.

See also

`interface`  
`interface-display`  
`interface-preserve-state`

## 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> <i>initargs</i>	An instance of a subclass of <i>interface</i> . Initargs for <i>interface</i> .
Values	<i>reusep</i>	A boolean
Description	The generic function <code>interface-reuse-p</code> returns a true value if <i>interface</i> is suitable for reuse with <i>initargs</i> .  <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.	
Notes	<code>interface-reuse-p</code> should not be confused with <code>reuse-interfaces-p</code> , which determines the global re-use state.	
See also	<code>interface-match-p</code> <code>locate-interface</code>	

**interface-toolbar-state***Function*

Signature	<code>interface-toolbar-state</code>	<i>interface</i> <i>key</i> => <i>value</i>  ( <code>setf interface-toolbar-state</code> ) <i>value</i> <i>interface</i> <i>key</i> => <i>value</i>
Arguments	<i>interface</i> <i>key</i> <i>value</i>	An instance of <i>interface</i> or a subclass. One of the <i>toolbar-states</i> plist keys. The value associated with the <i>toolbar-states</i> plist key.

Values	<i>value</i>	The value associated with the <i>toolbar-states</i> plist key.
Description	<p>The functions <code>interface-toolbar-state</code> and <code>(setf interface-toolbar-state)</code> read or change the properties of a toolbar that give information about its state. The user can also change these properties by customizing the toolbar, so you cannot assume that the value will be the same each time you read it.</p> <p><i>key</i> can be one of the following, with the corresponding value:</p> <p><b><code>:visible</code></b>      <i>visible</i> is true if the toolbar is visible and false if it is hidden. The default is true.</p> <p><b><code>:items</code></b>          <i>items</i> is a list of the names of the <i>toolbar-items</i> which are shown on the toolbar, in the order they are shown. The built-in names <code>:separator</code>, <code>:space</code> and <code>:flexible-space</code> represent various kinds of gap between items. On Microsoft Windows, an item can be a list of the form <code>(:titled-separator <i>title</i>)</code> which starts a dockable group of items that displays <i>title</i> when it is undocked. The default <i>items</i> includes all items in <i>toolbar-items</i>, with <code>:separator</code> between each <i>toolbar-component</i>.</p> <p><b><code>:display</code></b>        <i>display</i> is a keyword describing what is displayed for each item. It can be <code>:image</code> (just shows an image), <code>:title</code> (just shows the title), <code>:image-and-title</code> (shows both title and image) or <code>:image-and-title-horizontal</code> (shows title and image horizontally, only supported on GTK+). The default is platform-specific.</p>	

**:size**            *size* is a keyword describing the size of the items. It can be one of **:small**, **:normal** or **:large**. Some of these sizes might be the same as others. The default is platform-specific.

You can set all of the keys simultaneously by setting the **interface-toolbar-state** accessor or providing the *toolbar-states* **initarg**.

See also            **interface**  
**interface-customize-toolbar**

## interface-visible-p

*Function*

**Summary**            The predicate for whether the interface containing a pane is visible.

**Package**            **capi**

**Signature**            **interface-visible-p** *pane* => **visiblep**

**Arguments**            *pane*                    A CAPI pane.

**Values**                *visiblep*                A boolean.

**Description**            The function **interface-visible-p** returns **nil** if

1. *pane* 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's</i> 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>
Example	See the examples in the directory <code>examples/capi/layouts/</code> .
See also	<code>parse-layout-descriptor</code> <code>define-layout</code> <code>layout</code> <code>interface</code>

**invalidate-pane-constraints***Function*

Summary	Causes the resizing of a pane if its minimum and maximum size constraints have changed. It returns <code>t</code> if resizing was necessary.
Package	<code>capi</code>
Signature	<code>invalidate-pane-constraints</code> <i>pane</i>
Description	This function informs the CAPI that <i>pane</i> '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 <code>invalidate-pane-constraints</code> returns <code>t</code> .
See also	<code>get-constraints</code> <code>layout</code> <code>element</code> <code>define-layout</code>

**invoke-command***Function*

Summary	Invokes a command in the input model for a specified output pane.
Package	<code>capi</code>
Signature	<code>invoke-command</code> <i>command</i> <i>output-pane</i> &rest <i>event-args</i>
Description	This invokes the command in the input model for the given <i>output-pane</i> , with the translator being called to process the gesture information. To avoid the translation, use <code>invoke-untranslated-command</code> .

See also `invoke-untranslated-command`  
`define-command`  
`output-pane`

## **invoke-untranslated-command**

*Function*

**Summary** Invokes a command in the input model for a specified output pane, without the translator being called.

**Package** `capi`

**Signature** `invoke-untranslated-command command output-pane`  
`&rest event-args`

**Description** The function `invoke-untranslated-command` invokes the command in the input model for the given *output-pane*, without the translator being called to process the gesture information. To perform the translation, use `invoke-command`.

See also `invoke-command`  
`define-command`  
`output-pane`

## **item**

*Class*

**Summary** The class `item` groups together a title, some data and some callbacks into a single object for use in collections and choices.

**Package** `capi`

**Superclasses** `callbacks`  
`capi-object`

Subclasses	<pre> menu-item button item-pinboard-object popup-menu-button toolbar-button </pre>
Initargs	<pre> :collection  The collection in which item is displayed :data        The data associated with the item. :text        The text to appear in the item (or nil). :print-function               If <i>text</i> is nil, this is called to print the data. :selected    If <i>t</i> the item is selected. </pre>
Accessors	<pre> item-collection item-data item-text item-print-function item-selected </pre>
Description	<p>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.</p> <p>An item is printed in the collection by <code>print-collection-item</code>. By default this returns a string using <i>item's text</i> if specified, or else calls a print function on the item's <i>data</i>. The <i>print-function</i> will either be the one specified in the item, or else the <i>print-function</i> for its parent collection.</p> <p>The <i>selected</i> slot in an item is non-nil if the item is currently selected. The accessor <code>item-selected</code> is provided to access and to set this value.</p>
Example	<pre> (defun main-callback (data interface)   (capi:display-message "Main callback: ~S"     data))  (defun item-callback (data interface)   (capi:display-message "Item callback: ~S"     data)) </pre>

```
(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	<code>pinboard-object</code> <code>item</code>
Description	The <code>item-pinboard-object</code> 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 <code>print</code> function).
Example	<pre>(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))</pre>
See also	<code>image-pinboard-object</code> <code>pinboard-layout</code>

## labelled-arrow-pinboard-object

*Class*

Package	<code>capi</code>
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	<pre>(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))))</pre>
See also	<code>pinboard-layout</code>

## layout

*Class*

Summary A `layout` is a simple pane that positions one or more child panes within itself according to a layout policy.

Package	<code>capi</code>
Superclasses	<code>titled-object</code> <code>simple-pane</code>
Subclasses	<code>simple-layout</code> <code>grid-layout</code> <code>pinboard-layout</code> <code>switchable-layout</code>
Initargs	<p><code>:default</code> A flag to mark the default layout for an interface.</p> <p><code>:description</code> The list of the layout's children.</p> <p><code>:initial-focus</code> A child of the layout, or its <i>name</i>, specifying where the input focus should be, or <code>nil</code>.</p>
Accessors	<code>layout-description</code>
Description	<p>The layout's <i>description</i> is an abstract description of the children of the layout, and each layout defines its format. Generally, <i>description</i> is a list, each element of which is one of:</p> <ul style="list-style-type: none"> <li>• a pane</li> <li>• a slot name, where the name refers to a slot in the layout's interface containing a pane</li> <li>• a string, where the string gets converted to a <code>title-pane</code></li> </ul> <p>For <code>grid-layout</code> and its subclasses, elements of <i>description</i> can also be <code>nil</code>. See <code>grid-layout</code> for the interpretation of this value.</p> <p>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.</p> <p>A number of default layouts are provided which provide the majority of layout functionality that is needed. They are as follows:</p>

`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 `capi-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	<code>pinboard-object</code>								
Subclasses	<code>arrow-pinboard-object</code> <code>right-angle-line-pinboard-object</code>								
Initargs	<table> <tr> <td><code>:start-x</code></td> <td>The x coordinate of the start of the line.</td> </tr> <tr> <td><code>:start-y</code></td> <td>The y coordinate of the start of the line.</td> </tr> <tr> <td><code>:end-x</code></td> <td>The x coordinate of the end of the line.</td> </tr> <tr> <td><code>:end-y</code></td> <td>The y coordinate of the end of the line.</td> </tr> </table>	<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.
<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	<p><code>start-x</code>, <code>start-y</code>, <code>end-x</code> and <code>end-y</code> default to values computed from the <code>x</code>, <code>y</code>, <code>width</code> and <code>height</code>. They are used to compute the size of the object, and the proper value of <code>x</code> and <code>y</code>. Note that <code>width</code> and <code>height</code> may be larger, for example to accomodate the label in a <code>labelled-line-pinboard-object</code>, and the <code>x</code> and <code>y</code> are adjusted for that.</p> <p>To change the end points of the line, call <code>move-line</code>.</p> <p>A complementary class <code>right-angle-line-pinboard-object</code> is provided which draws a line around the edge of the pinboard object.</p>								
Example	<pre>(capi:contain  (make-instance   'capi:line-pinboard-object   :start-x 0 :end-x 100   :start-y 100 :end-y 0))</pre>								
See also	<code>move-line</code> <code>pinboard-layout</code>								

## line-pinboard-object-coordinates

*Function*

Summary Returns the coordinates of a `line-pinboard-object`.

Package `capi`

Signature	<code>line-pinboard-object-coordinates</code> <i>object</i> => <i>start-x</i> , <i>start-y</i> , <i>end-x</i> , <i>end-y</i>	
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.
Description	The function <code>line-pinboard-object-coordinates</code> returns the start and end coordinates of the <code>line-pinboard-object</code> <i>object</i> .	
See also	<code>move-line</code>	

## list-panel

*Class*

Summary	The class <code>list-panel</code> is a pane that can display a group of items and provides support for selecting items and performing actions on them.	
Package	<code>capi</code>	
Superclasses	<code>choice</code> <code>simple-pane</code> <code>sorted-object</code> <code>titled-object</code>	
Subclasses	<code>list-view</code> <code>multi-column-list-panel</code>	
Initargs	<code>:right-click-selection-behavior</code>	A keyword or <code>nil</code> . Controls the behavior on a right mouse button click.

`:color-function`

A function designator or `nil`. Controls item text color on Microsoft Windows and GTK+.

`:filter`

A boolean. The default value is `nil`.

The following `initargs` take effect only when *filter* is non-`nil`.

`:filter-automatic-p`

A boolean. The default value is `t`.

`:filter-callback`

A function designator or the keyword `:default`, which is the default value.

`:filter-change-callback-p`

A boolean.

`:filter-short-menu-text`

A boolean. The default value is `nil`.

`:filter-matches-title`

A string, `t` or `nil`.

`:filter-help-string`

A string, `t` or `nil`.

Accessors

`list-panel-right-click-selection-behavior`

Description

The class `list-panel` gains most of its behavior from `choice`, which is an abstract class that handles items and their selection. By default, a list panel has both horizontal and vertical scrollbars.

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 and GTK+. 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* is 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.

If *filter* is non-nil, the system automatically adds a `filtering-layout` above the list. The items in the `list-panel` are filtered by the value in the `filtering-layout`. Filtering displays only those items whose print representation matches the filter. (The print representation is the result of `print-collection-item`, and is what the user sees.) Only the items that match, or those that do not match if **Exclude** is set, are displayed in the `list-panel`.

Here filtering means mapping over the unfiltered items, collecting each item that matches the current setting in the filter, and then setting the items of the `list-panel` to the collected items.

For a `list-panel` with a filter, `collection-items` returns only the filtered items, and the selection (that is, the result of `choice-selection` and the argument to `(setf choice-selection)`) index into the filtered items.

Calling `(setf collection-items)` on a filtered `list-panel` sets an internal unfiltered list, and then clears the filtering so that all items are visible.

To get and set the unfiltered items, use the accessor `list-panel-unfiltered-items`. To access the filter-state, use `list-panel-filter-state`. To access both the unfiltered items and the filter simultaneously, which is especially useful when setting both of them at the same time, use `list-panel-items-and-filter`.

*filter-automatic-p* controls whether the filter automatically does the filtering whenever the text in the filter changes, and *filter-callback* defines the callback of the `filtering-layout`.

If *filter-automatic-p* is `t`, whenever a change occurs in the filter the list is refreshed against the new value in the filter. The *filter-callback* (if non-nil) is called with two arguments, the `filtering-layout` and the `list-panel` itself, when the user "confirms" (that is, she presses `Return` or clicks the **Confirm** button). If *filter-automatic-p* is false and *filter-callback* is `:default`, then the `filtering-layout` is given a callback that does the filtering when the user "confirms". If *filter-automatic-p* is false and *filter-callback* is non-nil, then no filtering is done explicitly, and it is the responsibility of the callback to do any filtering that is required.

*filter-matches-title* (default `t`) and *filter-help-string* (default `t`) are passed down to the filtering layout through the corresponding `filtering-layout` initargs:

*filter-matches-title*:`matches-title`

*filter-help-string* :`help-string`

See `filtering-layout` for a description of these initargs.

If *filter-short-menu-text* is true, the filter menu has a short title. For example if the filter is set for case-sensitive plain inclusive matching the short label is **PMC**. If *filter-short-menu-text* were false then this label would be **Filter:C**.

## Notes

If you use *filter*:

1. You should not rely on the `element-parent` of the `list-panel`, because it is implemented by wrapping some layouts around the `list-panel`.
2. The filter is actually a filtering layout, so it has the same interactive semantics as `filtering-layout`.

## 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/`.

This example illustrates the use of *color-function*:  
`examples/capi/applications/simple-symbol-browser.lisp`

See also `button-panel`

## list-panel-enabled

## Generic Function

Summary Gets or sets the enabled state of a `list-panel`.

Package `capi`

Signature	<code>list-panel-enabled</code> <i>list-panel</i> => <i>enabledp</i>
Signature	<code>(setf list-panel-enabled)</code> <i>enabledp list-panel</i> => <i>enabledp</i>
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-panel-filter-state

## Generic Function

Summary	Accesses the state of the filter in a filtered <code>list-panel</code> .
Signature	<code>list-panel-filter-state</code> <i>list-panel</i> => <i>filter-state</i> <code>(setf list-panel-filter-state)</code> <i>new-state list-panel</i>
Description	<p>The generic function <code>list-panel-filter-state</code> accesses the state of the filter in a filtered <code>list-panel</code> (that is, a <code>list-panel</code> created with <i>filter</i> <code>t</code>).</p> <p><code>list-panel-filter-state</code> returns the state of the filter in <i>list-panel</i>. The return value <i>filter-state</i> is the same type as the state that is used in <code>filtering-layout</code>.</p> <p><code>(setf list-panel-filter-state)</code> sets the filter in <i>list-panel</i>, filters the unfiltered items and displays those that match the <i>new-state</i>. The <i>new-state</i> has the same semantics as the <i>new-value</i> of <code>(setf filtering-layout-state)</code>. It can be a result</p>

of a call to `list-panel-filter-state` or to `filtering-layout-state` (on a `filtering-layout`), or a string (meaning plain match, case-insensitive), or `nil` (meaning match everything).

On an unfiltered `list-panel` `list-panel-filter-state` returns `nil`, and `(setf list-panel-filter-state)` does nothing.

See also `list-panel`  
`list-panel-unfiltered-items`  
`filtering-layout`

## list-panel-items-and-filter

*Function*

Summary      Accesses the unfiltered items and filter in a `list-panel`

Signature      `list-panel-items-and-filter` *list-panel*  
`(setf list-panel-items-and-filter ) (values items filter)`  
*list-panel*

The function `list-panel-items-and-filter` accesses the unfiltered items and the filter in the list panel *list-panel* simultaneously. It is especially useful for setting the filter and the items without flickering.

`list-panel-items-and-filter` returns the items and filter in *list-panel* as multiple values. It is equivalent to

```
(values (list-panel-unfiltered-items list-panel)
        (list-panel-filter-state list-panel))
```

but is more efficient.

`(setf list-panel-items-and-filter)` takes the items and filters as two values and sets them in *list-panel*:

```
(setf (list-panel-items-and-filter list-panel)
      (values new-items new-filter))
```

ends up in the same state as

```
(progn
  (setf (list-panel-unfiltered-items list-panel) new-items)
  (setf (list-panel-filter-state list-panel) new-filter))
```

but the latter form will filter the *new-items* with the old filter and display the result, and then filter the *new-items* again with the *new-filter*, whereas `(setf list-panel-items-and-filter)` filters the *new-items* just once, with the *new-filter*.

See also

- `list-panel`
- `list-panel-filter-state`
- `list-panel-unfiltered-items`

## list-panel-unfiltered-items

## Generic Function

**Summary**      Accesses the unfiltered items of a filtered `list-panel`.

**Signature**      `list-panel-unfiltered-items list-panel`  
`(setf list-panel-unfiltered-items) new-items list-panel`

**Description**      The generic function `list-panel-unfiltered-items` accesses the unfiltered items of a filtered `list-panel` (that is, a `list-panel` created with `:filter t`).

`list-panel-unfiltered-items` returns the unfiltered items of *list-panel* (that is all of them, as opposed to the accessor `collection-items`, which returns only those items that match the filter).

`(setf list-panel-unfiltered-items)` sets the items of *list-panel* without affecting the filter (as opposed to `(setf collection-items)` which resets the filter). The items are then filtered, and only those that match the filter are displayed.

`list-panel-unfiltered-items` behaves the same as `collection-items` when called on an unfiltered `list-panel`.

See also `list-panel`  
`list-panel-items-and-filter`  
`list-panel-filter-state`

## **list-view**

*Class*

Summary The list view pane is a choice that displays its items as icons and text in a number of formats.

**Note:** `list-view` is not implemented on Cocoa

Package `capi`

Superclasses `list-panel`

Initargs

- `:view` Specifies which view the list view pane shows. The default is `:icon`.
- `:subitem-function`  
Returns additional information to be displayed in report view.
- `:subitem-print-functions`  
Used in report view to print the additional information.
- `:image-function`  
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 `:a11`.

`: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.

`:normal`

Specifies an image-list object that contains the large item images. The *image-function* 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

`capi`

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 <code>*standard-output*</code> 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 <i>pane</i>
See also	<code>interactive-pane-execute-command</code>

## 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, as follows:

Microsoft Windows

`.cur` or `.ani` format.

Cocoa TIFF format.

GTK+ Any image format that `load-image` supports.

**Note:** The image can be of any dimension, but it will be clipped to what the server thinks is an appropriate size, 32x32 or 16x16. Using large images would waste space, because the image would still be in memory.

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 such as `:cocoa`, `:win32` or `:gtk` (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 these are not used on Microsoft Windows. Hotspot keyword arguments `:x-hot` and `:y-hot` are supported on Cocoa and GTK+ as in the example below. They specify the hotspot of the cursor. The values must be integers inside the image dimensions, that is they satisfy:

```
(and (> image-width x-hot -1)
 (> image-height y-hot -1))
```

On GTK+ the library specific arguments also include the keywords `:transparent-color-index` and `:type`, which are passed to `read-external-image`. Note that supplying the *transparent-color-index* allows making a useful cursor with a simple format image file which does not have transparency.

#### Example

This example loads a standard Microsoft Windows cursor file:

```
(setq curl (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 "3dwns")
                   (: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      `capi`

Signature     `load-sound source &key owner => sound`

Arguments    *source*            A pathname designator or an array returned by `read-sound-file`.

*owner*            A CAPI interface, or `nil`.

Values        *sound*            An array of element type `(unsigned-byte 8)`.

Description   The function `load-sound` converts *source* into a loaded sound which can be played by `play-sound`.

*source* can be a pathname designator or an array returned by `read-sound-file`.

*owner* should be a CAPI interface object, or `nil` which means that the sound's owner is the current top level interface.



If there is no match, then `locate-interface` finds the first of these which can be reused for *initargs*, by `interface-reuse-p`. This reusable interface is reinitialized by `reinitialize-interface` and returned.

*no-busy-interface* controls the use of the busy cursor during reinitializing of a reusable interface. If *no-busy-interface* is `nil`, then this interface has the busy cursor during reinitialization. If *no-busy-interface* is true, then there is no busy cursor.

If no matching or reusable interface is found, or if global interface re-use is disabled by `(setf reuse-interfaces-p)`, then `locate-interface` returns `nil`.

See also `collect-interfaces`  
`interface-match-p`  
`interface-reuse-p`  
`reuse-interfaces-p`

## lower-interface

## Function

Summary	The <code>lower-interface</code> function pushes the window containing a specified pane to the back of the screen.
Package	<code>capi</code>
Signature	<code>lower-interface</code> <i>pane</i>
Description	This pushes the window containing <i>pane</i> to the back of the screen. To bring it back use <code>raise-interface</code> , and to iconify it use <code>hide-interface</code> .
See also	<code>hide-interface</code> <code>interface</code> <code>lower-interface</code> <code>raise-interface</code> <code>quit-interface</code>

## 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        `controller`        A docking layout controller.

Description   The function `make-docking-layout-controller` returns a docking layout controller object for use as the *controller* initarg in `docking-layout`.

Layouts which share a docking layout controller are known as a Docking Group. See `docking-layout` for information about Docking Groups.

See also       `docking-layout`

## **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       `capi`

Signature     `make-foreign-owned-interface &key handle name => interface`

Arguments    `handle`            A Microsoft Windows hwnd.

`name`            A string naming *interface*.

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 <code>hwnd</code>) 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 window handle will depend on the language and the way windows are represented, so you should consult the appropriate documentation.</p> <p><i>name</i> becomes the name of <i>interface</i>, and has no other meaning.</p> <p><code>make-foreign-owned-interface</code> is implemented only on Microsoft Windows.</p>
Example	<p>This example shows how a CAPI window can be the owner of a dialog in another LispWorks image.</p> <p>Start LispWorks for Windows.</p> <ol style="list-style-type: none"> <li>1. In the Listener, do <b>Tools &gt; Interface &gt; Listen</b>. This puts the Listener interface in the value of <code>*</code>.</li> <li>2. In the Listener enter <code>(capi:simple-pane-handle *)</code>. The returned value is the window handle, it should be an integer. Denote this value by <i>hwnd</i>.</li> </ol> <p>Start another LispWorks for Windows image (do not quit the first image). In the Listener of this second LispWorks image:</p> <ol style="list-style-type: none"> <li>1. Enter <code>(setq foi (capi:make-foreign-owned-interface :handle hwnd))</code>.</li> <li>2. Enter <code>(capi:prompt-for-color "Color?" :owner foi)</code>.</li> </ol>

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 &amp;key <i>image-count width height id</i> =&gt; <i>image-set</i></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>	
Example	<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 `image-set`  
`make-general-image-set`

## make-image-locator

*Function*

Summary Creates an image locator object to use with toolbars, list views and tree views.

Package `capi`

Signature `make-image-locator &key image-set index`

Description The function `make-image-locator` creates an image locator object for use with toolbars, list views, and tree views. It is used to specify a single sub-image from a larger image that contains many images side by side. It is also useful for accessing some images that can only be specified by means of image sets.

See also `image-set`

## make-menu-for-pane

*Function*

Summary Makes a menu or a menu-component for a pane.

Package `capi`

Signature `make-menu-for-pane pane items  
&key title menu-name component-p => menu`

Arguments `pane` A pane.  
`items` A list of menu-objects.  
`title` A string or nil.  
`menu-name` A string or nil.

	<i>component-p</i>	A boolean.
Values	<i>menu</i>	A menu or a menu-component.
Description	<p>The function <code>make-menu-for-pane</code> makes a menu or a menu-component for the pane <i>pane</i> with the items specified by <i>items</i>. <i>items</i> should be a list in which each element is a menu-item, menu-component or menu.</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-menu-for-pane</code> creates a menu-component rather than a menu. The default value of <i>component-p</i> is <code>nil</code>.</p> <p><i>menu</i> is set up so that by default each callback inside it is done on the pane <i>pane</i> itself. This is the useful feature of <code>make-menu-for-pane</code> because it avoids the need to set up items to do their callbacks on <i>pane</i> explicitly.</p> <p>Note that this is merely the default behavior. You can specify different callback behavior on a per-item basis, using <i>setup-callback-argument</i> and <i>callback-data-function</i> (see <code>menu-object</code>), <i>callback-type</i> (see <code>callbacks</code>) and <i>data</i> for <code>menu-item</code> (see <code>item</code>).</p>	
See also	<p><code>make-pane-popup-menu</code>  <code>pane-popup-menu-items</code></p>	

## make-pane-popup-menu

## Generic Function

Summary	Generates a popup menu or menu-component.	
Package	<code>capi</code>	
Signature	<code>make-pane-popup-menu</code> <i>pane interface</i> <i>&amp;key title menu-name component-p =&gt; 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 menu or a menu-component.
Description	The generic function <code>make-pane-popup-menu</code> generates a popup menu for <i>pane</i> .	
	<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> ).	
	<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> .	
	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> .	
Example	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.	
	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.	

```

(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 &amp;key <i>image-count</i> <i>width</i>  <i>height</i> <i>base-image-set</i> =&gt; <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 or removes one or more `pinboard-objects` on a `pinboard`.

**Package** `capi`

**Signature** `manipulate-pinboard` *pinboard-layout* *pinboard-object* *action* &key *position*

**Arguments**

- pinboard-layout* A `pinboard-layout`.
- pinboard-object* A `pinboard-object` to be added, or (with *action* `:add-many`) a list of `pinboard-objects` to be added.  
 With *action* `:delete-if`, *pinboard-object* can also be a function of one argument, for multiple deletion.
- 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 p1
      (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>
Arguments	<i>collection</i> A collection. <i>function</i> A function designator for a function of one argument. <i>collect-results-p</i> A generalized boolean.
Description	Calls <i>function</i> on each item in the <i>collection</i> by calling the collection's <i>items-map-function</i> . If <i>collect-results-p</i> is true, the results of these calls are 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.

*function*            A function of one argument.  
*visible*             A boolean. The default value is `nil`.  
*test*                 A function of one argument, or `nil`. The default is `nil`.  
*reverse*             A boolean. The default value is `nil`.

Description        The function `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> &key <i>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> .
	<i>leaf-only</i>	A generalized boolean. The default value is <code>nil</code> .
Description	<p>The function <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-nil, 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-nil, the order in which the children are processed is reversed.</p> <p>If <i>leaf-only</i> is true, then <i>function</i> is applied only to those panes which do not have children.</p>	
See also	<code>map-pane-children</code> <code>pane-descendant-child-with-focus</code>	

## map-typeout

*Function*

Package	<code>capi</code>
Signature	<code>map-typeout <i>pane</i> &amp;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	<code>unmap-typeout</code> <code>with-random-typeout</code> <code>collector-pane</code>

## \*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>#\&amp;</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	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> .

The simplest way of providing items to a menu is to pass them as the argument *items*, but if you need to compute the items dynamically you should provide the setup callback *items-function*. This function should return a list of menu items for the new menu. By default *items-function* is called on the menu's interface, but a different argument can be specified using the `menu-object` initarg *setup-callback-argument*.

**Note:** *items-function* is called before the menu is raised (in order to initialize accelerators) and in particular it may be called before the interface is created. Therefore *items-function*, if you supply it, should work at this early stage.

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:

`nil`                    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.

Notes

1. On Cocoa and GTK+, 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. You can test programmatically whether menus with images are supported with `pane-supports-menus-with-images`.
2. 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.
3. 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`.

4. By default Microsoft Windows hides mnemonics when the user is not using the keyboard. In Windows XP (and later) a system preference controls this:

**Display > Appearance > Effects > Hide underlined letters...**

Example

```
(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 is an example showing how to make a menu with images in `examples/capi/elements/menu-with-images.lisp`.

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

See also

```
display-popup-menu
menu-component
menu-item
menu-object
ole-control-add-verbs
pane-supports-menus-with-images
popup-menu-button
```

## 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	<code>capi</code>
Superclasses	<code>choice</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 setup callback function to dynamically compute the items.</p> <p><code>:selection-function</code> A setup callback function to dynamically compute the selection.</p> <p><code>:selected-item-function</code> A setup callback function to dynamically compute the selected item.</p> <p><code>:selected-items-function</code> A setup callback function to dynamically compute the selected items.</p>
Description	<p>Because <code>menu-component</code> is a choice, the component can have <i>interaction</i> <code>:no-selection</code>, <code>:single-selection</code> or <code>:multiple-selection</code> (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).</p> <p>Note that it is not appropriate to have menu components or submenus inside <code>:single-selection</code> and <code>:multiple-selection</code> components, but it is OK in <code>:no-selection</code> components.</p> <p><i>items</i> and <i>items-function</i> behave as in <code>menu</code>.</p>

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** `:accelerator`

A character, string or plist, or the keyword `:default`.

`:alternative`

A generalized boolean.

`:help-key`

An object used for lookup of help. Default value `t`.

`:mnemonic`

A character, integer or symbol specifying a mnemonic for the menu item.

`:mnemonic-escape`

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

`:mnemonic-title`

A string specifying the text and a mnemonic.

`:selected-function`

A setup callback determining whether the item is selected.

	<p><code>:enabled-function-for-dialog</code></p> <p><code>nil</code>, <code>t</code>, <code>:same-as-normal</code> or a function designator. Determines enabled state when a dialog is on screen.</p>
Readers	<code>help-key</code>
Description	<p>The text displayed in the menu item is the contents of the <i>text</i> slot, or the contents of the <i>title</i> slot, otherwise it is the result of applying the <i>print-function</i> to the <i>data</i>.</p> <p>If <i>selected-function</i> is non-<code>nil</code> it should be a function which is called before the <code>menu-item</code> is displayed and which determines whether or not the <code>menu-item</code> is selected. By default <i>selected-function</i> is called on the interface of the <code>menu-item</code>, but this argument can be changed by passing the <code>menu-object</code> initarg <i>setup-callback-argument</i>. The default value of <i>selected-function</i> is <code>nil</code>.</p> <p>Callbacks are made in response to a user gesture on a <code>menu-item</code>. The <i>callback-type</i> (see <code>callbacks</code>), <i>callback</i> and <i>callback-data-function</i> (see <code>menu-object</code>) are found by looking for a non-<code>nil</code> value, first in the <code>menu-item</code>, then the <code>menu-component</code> (if any) and finally the <code>menu</code>. This allows a whole menu to have, for example, <i>callback-type</i> <code>:data</code> without having to specify this in each item. Some items could override this by having their <i>callback-type</i> slot non-<code>nil</code> if needed.</p> <p>To specify a mnemonic in the menu item, you can use the initarg <code>:mnemonic</code>, or the initargs <code>:mnemonic-title</code> and <code>:mnemonic-escape</code>. These initargs are all interpreted just as in <code>menu</code>.</p> <p>A menu item should not be used more than once in more than one place at a time.</p> <p><i>help-key</i> is interpreted as described for <code>element</code>.</p> <p><i>accelerator</i> can be a character or string specifying a key gesture which will be the accelerator for the menu item.</p>

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*.

*enabled-function-for-dialog* determines whether the item is enabled when a dialog is on the screen. Items in the menu bar menus and sub-menus are disabled by default while a dialog is on the screen on top of the active window. You can override this by specifying *enabled-function-for-dialog*. The value can be one of:

<code>t</code>	The item is enabled whenever there is a dialog.
<code>nil</code>	The item is disabled whenever there is a dialog.
<code>:same-as-normal</code>	Do the same as when there is no dialog. This depends on the <i>enabled-function</i> (see <code>menu-object</code> ).
A function	A function that is called instead of the <i>enabled-function</i> to decide if the item should be enabled. It is called with one argument, by the default the menu interface, which can be overridden by the <code>initarg :setup-call-back-argument</code> (see <code>menu-object</code> for details).

The default value of *enabled-function-for-dialog* is `nil`.

Notes Some accelerators do not work on some platforms because they have other standard meanings, for example on Microsoft Windows **F1** always invokes the *help-callback*.

On X11/Motif the accelerators of alternative items do not work.

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 "accelerator-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 <code>menu-object</code> is the superclass of all menu objects, and provides functionality for handling generic aspects of menus, menu components and menu items.
Package	<code>capi</code>
Superclasses	<code>callbacks</code>
Subclasses	<code>titled-menu-object</code>
Initargs	<code>:popup-callback</code> Callback before the menu appears.  <code>:enabled-function</code> Returns true if the menu is enabled.  <code>:enabled-slot</code> The object is enabled if the slot is non-nil.  <code>:callback</code> The selection callback for the object.  <code>:callback-data-function</code> A function to return data for the callback.  <code>:setup-callback-argument</code> If non-nil, specifies the argument to the setup callbacks (listed below) that are used to set up the <code>menu-object</code> .
Accessors	<code>menu-popup-callback</code>
Readers	<code>menu-object-enabled</code>
Description	When the menu object is about to appear on the screen, the CAPI does the following:

1. The setup callback *items-function* (if there is one) is called and the result is used to set the items, for `menu` and `menu-component`. The argument passed to *items-function* is the same as for the other setup callbacks (see below).
2. The *popup-callback* (if there is one) is called and can make arbitrary changes to that object. The *popup-callback* is always called with the menu object, regardless of the value of *setup-callback-argument*.
3. The other setup callbacks are called to set up the selection, enabled state and title. These setup callbacks include *enabled-function* for all `menu-object`s and *title-function* for all `titled-menu-object`s. The additional setup callbacks for `menu-component` are *selection-function*, *selected-item-function*, and *selected-items-function*. `menu-item` has the additional setup callback *selected-function*.

By default *setup-callback-argument* is `nil`, which means that each of the setup callbacks is called on the interface of the `menu-object`. If *setup-callback-argument* is non-`nil`, 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.

Note that *enabled-slot* is a short-hand means of creating an *enabled-function* which checks the value of a slot in the menu object's interface.

The enabled state of a `menu-object` is computed each time the menu is displayed, using *enabled-function* or *enabled-slot*. Therefore the accessor `menu-object-enabled` is only useful as a reader.

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> <code>initarg</code> .
Package	<code>capi</code>

Superclasses	<code>title-pane</code>
Description	The class <code>message-pane</code> is used to implement the message decoration on subclasses of <code>titled-object</code> .  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".
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	The <code>modify-editor-pane-buffer</code> function modifies the <code>editor-pane</code> <i>pane</i> according to the keyword arguments.  The argument <i>contents</i> (if non-nil) supplies a new string to place in the buffer.  <i>flag</i> , if given, sets the flag slot of the editor buffer, which is used to mark it for various specialized uses.  If <i>fill</i> is non-nil 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.  <i>fixed-fill</i> defaults to <code>nil</code> .
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 line-pinboard-object start-x start-y end-x end-y &amp;key redisplay</code>
Arguments	<p><i>line-pinboard-object</i></p> <p>An instance of <code>line-pinboard-object</code> or a subclass.</p> <p><i>start-x</i>           The x coordinate of the start of the line.</p> <p><i>start-y</i>           The y coordinate of the start of the line.</p> <p><i>end-x</i>             The x coordinate of the end of the line.</p> <p><i>end-y</i>             The y coordinate of the end of the line.</p> <p><i>redisplay</i>         A boolean.</p>
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:** similiar 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" "Pumkpin"))))
```

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" "Pumkpin"))))
```

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`

## non-focus-list-interface

*Class*

Summary	Created (and destroyed) only by <code>prompt-with-list-non-focus</code> and <code>text-input-pane-in-place-complete</code> .
Superclasses	<code>interface</code>
Description	The class <code>non-focus-list-interface</code> is the class of interface created and destroyed only by <code>prompt-with-list-non-focus</code> and <code>text-input-pane-in-place-complete</code> . Do not instantiate this class directly.
See also	<code>prompt-with-list-non-focus</code> <code>text-input-pane-in-place-complete</code>

## non-focus-list-toggle-enable-filter

*Function*

Summary	Toggles the enabled state of the filter.
Signature	<code>non-focus-list-toggle-enable-filter</code> <i><b>non-focus-list-interface</b></i>
Arguments	<i>non-focus-interface</i> A <code>non-focus-list-interface</code> .
Description	The function <code>non-focus-toggle-enable-filter</code> toggles the enabled state of the filter in a non-focus list created by <code>prompt-with-list-non-focus</code> OR <code>text-input-pane-in-place-complete</code> . It has no effect if the filter is off. It is used as the callback of the <i>filtering-toggle</i> .
See also	<code>prompt-with-list-non-focus</code>

## non-focus-list-toggle-filter

### non-focus-list-add-filter

### non-focus-list-remove-filter

*Functions*

Summary	Add or remove the filter in a non-focus list.
Signature	<pre>non-focus-list-toggle-filter <i>non-focus-list-interface</i> non-focus-list-add-filter    <i>non-focus-list-interface</i> non-focus-list-remove-filter <i>non-focus-list-interface</i></pre>
Arguments	<p><i>non-focus-interface</i></p> <p>A <i>non-focus-list-interface</i>.</p>
Description	<p>These functions add or remove the filter in a non-focus list.</p> <p><i>non-focus-list-toggle-filter</i> calls <i>non-focus-list-add-filter</i> if the filter is off, otherwise it calls <i>non-focus-list-remove-filter</i> (it is used as the callabck for the <i>filtering-gesture</i>).</p> <p><i>non-focus-list-add-filter</i> adds a filter is it is not already on, resets the text in it to empty string, and enables it.</p> <p><i>non-focus-list-remove-filter</i> removes the filter if it is on.</p>
See also	<i>prompt-with-list-non-focus</i>

## non-focus-maybe-capture-gesture

*Generic Function*

Summary	Maybe capture a gesture by the non-focus-interface.
Signature	<pre>non-focus-maybe-capture-gesture <i>non-focus-interface</i> <i>gesture</i> =&gt; <i>result</i></pre>
Arguments	<p><i>non-focus-interface</i></p> <p>A <i>non-focus-list-interface</i>.</p>

	<i>gesture</i>	A gesture specifier.
Values	<i>result</i>	A generalized boolean.
Method Signature	<code>non-focus-maybe-capture-gesture (non-focus-interface non-focus-list-interface) <i>gesture</i></code>	
Description	<p>The generic function <code>non-focus-maybe-capture-gesture</code> needs to return non-nil if the gesture <i>gesture</i> was captured, which means it should not be processed any more, or nil if <i>gesture</i> was not captured.</p> <p><i>gesture</i> should be a gesture specifier, which is an object that can be coerced to a Gesture Spec by <code>sys:coerce-to-gesture-spec</code>.</p> <p>The method on <code>non-focus-list-interface</code> does the following:</p> <ol style="list-style-type: none"> <li>1. If the gesture is <code>Escape</code> it calls <code>non-focus-terminate</code> on the non-focus window.</li> <li>2. It checks whether the gesture matches any of the gestures in the <i>gesture-callbacks</i> of the window. The gesture callbacks are either explicitly defined using the initargs <code>:gesture-callbacks</code> or <code>:add-gesture-callbacks</code>, or implicitly. By default, all the gestures that are used in in-place completion (see "In-place completion" in the <i>Lisp-Works CAPI User Guide</i>) are defined implicitly. These include <code>Up</code>, <code>Down</code>, <code>PageUp</code>, <code>PageDown</code> (selection in the list panel), <code>Return</code> (action), <code>Control+Return</code> and <code>Control+Shift+Return</code> (control of the filter). The implicitly defined gestures are affected by <i>gesture-callbacks</i>, <i>filtering-gesture</i> and <i>filtering-toggle</i>.</li> </ol> <p>If a match is found, it is invoked as described for <i>gesture-callbacks</i> in <code>prompt-with-list-non-focus</code>.</p> <ol style="list-style-type: none"> <li>3. If filtering is enabled, it checks if the gesture is captured by the filter. A gesture is captured by the filter if it is:</li> </ol>	

A plain graphic character.

It is inserted to the filter

**Backspace**

The last character in the filter is deleted

One of the gestures which update the state of the filter (by default `Control+Shift+R`, `Control+Shift+E`, `Control+Shift+C`)

The state of the filter is updated.

In any case, where a gesture is captured by the filter the list panel is updated.

If the gesture is captured by one of the possibilities above, the method returns `t`, otherwise it returns `nil`.

See also

`non-focus-terminate`  
`prompt-with-list-non-focus`

**non-focus-terminate**

*Generic Function*

Summary

Terminates the non-focus interface.

Signatures

`non-focus-terminate` *non-focus-interface*

Method Signature

`non-focus-terminate` (*non-focus-interface* non-focus-list-interface)

Description

The generic function `non-focus-terminate` closes the non-focus interface.

It has no return value.

The method terminates a `non-focus-list-interface`. It destroys the interface in the correct process.

See also

`prompt-with-list-non-focus`

## non-focus-update

*Generic Function*

Summary	Updates the non-focus-interface.
Signature	<code>non-focus-update</code> <i>non-focus-interface</i>
Method Signature	<code>non-focus-update</code> ( <i>non-focus-interface</i> non-focus-list-interface)
Description	<p>The generic function <code>non-focus-update</code> updates the non-focus-interface.</p> <p>It has no return value.</p> <p>The method on <code>non-focus-list-interface</code> needs to be invoked in the process in which the <i>list-updater</i> that was passed to <code>prompt-with-list-non-focus</code> is expecting to run.</p> <p>It invokes the <i>list-updater</i> without arguments, and then updates the non-focus-interface with result. See the description of <i>list-updater</i> in <code>prompt-with-list-non-focus</code>.</p> <p>Note that if <i>list-updater</i> returns <code>:destroy</code>, this invokes <code>non-focus-terminate</code> on the interface.</p>
See also	<code>prompt-with-list-non-focus</code> <code>non-focus-terminate</code>

## prompt-with-list-non-focus

*Function*

Summary	Raises a non-focus window.
Signature	<code>prompt-with-list-non-focus</code> <i>items</i> &key <i>owner</i> <i>x</i> <i>y</i> <i>choice-class</i> <i>vertical-scroll</i> <i>print-function</i> <i>selection</i> <i>selected-item</i> <i>visible-items</i> <i>selection-callback</i> <i>action-callback</i> <i>destroy-callback</i> <i>list-updater</i> <i>gesture-callbacks</i> <i>add-gesture-callbacks</i> <i>alternative-y</i> <i>alternative-x</i> <i>alternative-bottom</i> <i>alternative-right</i> <i>widget-name</i> <i>filtering-gesture</i> <i>filtering-toggle</i> &allow-other-keys => <i>interface</i>

Arguments	<i>owner</i>	A displayed CAPI pane.
	<i>x,y</i>	Integers.
	<i>alternative-x, alternative-y</i>	Integers.
	<i>alternative-bottom, alternative-right</i>	Integers or <code>t</code> .
	<i>choice-class</i>	A subclass of <code>list-panel</code> .
	<i>selection</i>	An integer.
	<i>selected-item</i>	An item.
	<i>visible-items</i>	A positive integer.
	<i>vertical-scroll</i>	A boolean.
	<i>print-function</i>	A function designator or <code>nil</code> .
	<i>selection-callback</i>	A function designator or <code>nil</code> .
	<i>action-callback</i>	A function designator or <code>nil</code> .
	<i>destroy-callback</i>	A function designator or <code>nil</code> .
	<i>list-updater</i>	A function designator or <code>nil</code> .
	<i>gesture-callbacks</i>	A list of pairs of the form ( <i>gesture</i> . <i>callback</i> ).
	<i>add-gesture-callbacks</i>	A list of pairs of the form ( <i>gesture</i> . <i>callback</i> ).
<i>filtering-gesture</i>	A Gesture Spec.	
<i>filtering-toggle</i>	A Gesture Spec.	
<i>widget-name</i>	A string.	
Values	<i>interface</i>	A <code>non-focus-list-interface</code> , or <code>nil</code> .

## Description

The function `prompt-with-list-non-focus` raises a non-focus window, displaying the items *items* in a list of class *choice-class*, which should be `list-panel` or a subclass.

The non-focus window does not take the input focus, and hence does not see any keyboard input unless this is passed to it by `non-focus-maybe-capture-gesture`. It responds to mouse gestures.

Note that even moving the selection in the list vertically in response to the arrow keys cannot happen without `non-focus-maybe-capture-gesture`.

*owner* is required, and must be a CAPI pane visible on the screen. The position of the non-focus window is determined relative to *owner*, and the callbacks are invoked in the process of *owner*.

*x* and *y* are required pixel coordinates with respect to *owner* of the top left corner of the non-focus window.

*alternative-bottom*, *alternative-right*, *alternative-x* and *alternative-y* specify alternative locations for use when positioning the window at *x* or *y* would cause it to be off the screen. If *alternative-bottom* or *alternative-right* are specified, they specify alternative bottom or alternative right. For example, both Editor completion and `text-input-pane` completion specify a *y* coordinate below the text, and *alternative-bottom* above the text.

*alternative-bottom* and *alternative-right* can also take the special value `t`, which denotes the *height* or *width* of the `screen`.

*alternative-x* and *alternative-y* can be used to specify alternative *x* and alternative *y*. *alternative-bottom* overrides *alternative-y* and *alternative-right* overrides *alternative-x*.

The default value of *choice-class* is `list-panel`.

*selection* or *selected-item* can be used to specify the initially selected item in the list. If neither of these initargs is supplied, the first item is selected.

*visible-items* specifies the height of the list panel when the filter is not visible. The default value of *visible-items* is 20.

*vertical-scroll* is supplied to `c1:make-instance` when making the list. The default value of *vertical-scroll* is `t`.

*print-function* is also supplied to `c1:make-instance` when making the list. The default value of *print-function* is `nil`.

*selection-callback*, if non-`nil`, should be a function of two arguments, the selected item and the non-focus interface. *selection-callback* is called (in the process of *owner*) when an item is selected in the list panel. Note that *callback-type* does not affect the arguments passed to *selection-callback*.

*action-callback*, if non-`nil`, should also be a function of two arguments, the selected item and the non-focus interface. *action-callback* is called (in the process of *owner*) when an item is double-clicked in the list panel, or when `Return` is passed to `non-focus-maybe-capture-gesture` (by default, see *gesture-callbacks*). Note that *callback-type* does not affect the arguments passed to *action-callback*.

*destroy-callback*, if non-`nil`, should be a function of one argument, the non-focus window (a CAPI interface). *destroy-callback* is called when the non-focus window is destroyed. It is invoked in the process of *owner*.

*list-updater*, if non-`nil`, should be a function with signature

```
list-updater => result
```

*list-updater* is called in the process of *owner* whenever `non-focus-update` is called. *result* must be a list of items to put into the list panel, or one of the special values `t` (meaning no effect) and `:destroy` (meaning destroy the non-focus window).

*gesture-callbacks* and *add-gesture-callbacks* define gesture callbacks which the non-focus window can "capture" (when `non-focus-maybe-capture-gesture` is called). *gesture-callbacks* and *add-gesture-callbacks* should both be a list of pairs of the

form (*gesture* . *callback*). Each *gesture* must be a gesture specifier, that is an object that `sys:coerce-to-gesture-spec` can coerce to a Gesture Spec. Each *callback* is either a callable (symbol or function) which takes one argument, the non-focus window, or a list of the form (*function arguments*). Note that when it is a list, the window is not automatically passed to the function *function* amongst the arguments *arguments*. The gesture callbacks are used only when `non-focus-maybe-capture-gesture` is called.

*add-gesture-callbacks* adds more gesture callbacks to those that are implicitly defined for controlling the list panel (see `non-focus-maybe-capture-gesture`). *gesture-callbacks*, if supplied, replaces the gesture callbacks that are implicitly defined for the list panel. In both cases, a gesture callback that is defined explicitly overrides any implicitly define gesture callback.

*filtering-gesture* defines whether it is possible for the user to add a filter to the non-focus window with a keyboard gesture, and defines that gesture. The gesture is actually a toggle: it destroys a filter that is on, and adds a filter when none is present. When the filter is added, its text is reset and it is always enabled, that is it captures characters and `Backspace`. While the filter is visible, the list panel displays only items that match the filter. The default value of *filtering-gesture* is a Gesture Spec matching `Control+Return`.

*filtering-toggle* defines whether it is possible for the user to disable/enable the filter with a keyboard gesture, and defines that gesture. When a filter is visible and enabled, the non-focus window captures characters and `Backspace` (when `non-focus-maybe-capture-gesture` is called) and passes them to the filter. When the filter is visible and disabled, characters and `Backspace` are captured. The default value of *filtering-toggle* is a Gesture Spec matching `Control+Shift+Return`.

*widget-name* has an effect only on Motif. It defines the widget name of the interface, which can then be used to define resources specific to the non-focus window. Note that the non-focus completers in `editor-pane` and `text-input-pane` use the default *widget-name* which is `"non-focus-list-prompter"`, so defining resources for `non-focus-list-prompter` will affect them.

If *items* is `nil`, `prompt-with-list-non-focus` returns `nil` without doing anything. Otherwise, it raises the non-focus window and returns the interface, which is of class `non-focus-list-interface`.

The non-focus window is "passive", because it does not see keyboard input. It is the responsibility of the caller to pass any keyboard input that the non-focus window needs to process to the window, by using `non-focus-maybe-capture-gesture`. In general, that should be all keyboard gestures, and `non-focus-maybe-capture-gesture` decides which gestures it wants to process.

The caller can also use `non-focus-terminate`, `non-focus-update`, `non-focus-list-toggle-filter`, `non-focus-list-add-filter`, `non-focus-list-remove-filter` and `non-focus-list-toggle-enable-filter` to control the non-focus window.

See also

```
list-panel
non-focus-terminate
non-focus-update
non-focus-list-toggle-filter
non-focus-list-toggle-enable-filter
non-focus-maybe-capture-gesture
```

## ole-control-add-verbs

*Function*

Summary Adds to the menu entries for the "verbs" that a component in an `ole-control-pane` 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><b>Note:</b> 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</code> <i>pane</i>

Arguments	<i>pane</i> 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</code> <i>pane</i> .  <b>Note:</b> 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.  <code>:destroy-callback</code> A function called just before the pane is destroyed.
Readers	<code>ole-control-component-pane</code>

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> <p>Normally the program does not need to do anything else with the <code>ole-control-doc</code>. It acts in response to resizing of the window and method calls from the object on the <code>IOleInPlaceUIWindow</code> interface.</p> <p><b>Note:</b> <code>ole-control-doc</code> is implemented only in LispWorks for Windows. Load the functionality by <code>(require "embed")</code>.</p> <p><b>Note:</b> even though it is a subclass of <code>pinboard-layout</code>, normally you should not use the <code>pinboard-layout</code> functionality when using <code>ole-control-doc</code>.</p>

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 `examples/com/ole/simple-container/doc-viewer-pair.lisp`

See also `ole-control-insert-object`  
`ole-control-pane`

## ole-control-i-dispatch

*Function*

Summary Returns the `com:i-dispatch` of the component of an `ole-control-pane`.

Signature `ole-control-i-dispatch pane => result`

Arguments *pane* An `ole-control-pane`.

Values *result* A `com:i-dispatch` or `nil`.

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 <code>ole-control-pane</code> .
Signature	<code>ole-control-insert-object</code> <i>pane</i>
Arguments	<i>pane</i> An <code>ole-control-pane</code> .
Description	<p>The function <code>ole-control-insert-object</code> prompts the user for a document using the Microsoft Windows function <code>OleUIInsertObject</code>.</p> <p>When the user specifies a document in the dialog presented, <code>ole-control-insert-object</code> embeds this document in the <code>ole-control-pane</code> <i>pane</i>.</p> <p><b>Note:</b> 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>

## 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</code> <i>pane</i> => <i>result</i>
Arguments	<i>pane</i> An <code>ole-control-pane</code> .
Values	<i>result</i> A <code>com:i-ole-object</code> or <code>nil</code> .



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 `capi`

Superclasses `com:simple-i-dispatch`

Initargs `:ole-control-pane`

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><code>ole-control-pane</code> 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><b>Note:</b> <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	<p><code>attach-simple-sink</code></p> <p><code>ole-control-pane</code></p>

## ole-control-user-component

*Function*

Summary	Gets and sets the <i>user-component</i> of an <code>ole-control-pane</code> .
Signature	<pre>ole-control-user-component <i>pane</i> =&gt; <i>user-component</i> (setf ole-control-user-component) <i>user-component</i> <i>pane</i> =&gt; <i>user-component</i></pre>
Arguments	<p><i>pane</i>            An <code>ole-control-pane</code>.</p> <p><i>user-component</i>   A COM interface pointer.</p>
Description	<p>The function <code>ole-control-user-component</code> gets and sets the <i>user-component</i> of the <code>ole-control-pane</code> <i>pane</i>.</p> <p><i>user-component</i> (if non-nil) is a COM interface pointer of an object that supports the <code>I-OLE-OBJECT</code> interface, and has been opened and initialized and is ready to be displayed. This is typically created by calling <code>OleCreate</code>, <code>OleCreateFromFile</code>, <code>OleCreateFromData</code> or <code>OleLoad</code> with <i>pClientSite</i> null.</p>

The *user-component* is closed and released by the `ole-control-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.

*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*).

If *enabled-positions* is `:all` 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 `:all`.

#### Notes

1. `:image-function` is currently only implemented for Microsoft Windows and Cocoa.
2. On Motif, the separator is represented simply as a blank item between the other items.
3. On Motif and GTK+ versions older than 2.12, 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 are further examples in the files `examples/capi/choice/option-pane.lisp` and `examples/capi/choice/option-pane-with-images.lisp`.

## output-pane

*Class*

Summary	An output pane is a pane whose display and input behavior can be controlled by the programmer.
Package	<code>capi</code>
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

- :display-callback**  
A function that knows how to redisplay the pane.
- :input-model** A list of input specifications, otherwise known as a command table.
- :scroll-callback**  
A function called when the pane is scrolled, or `nil`. The default is `nil`.
- :pane-can-scroll**  
A generalized boolean specifying whether the pane itself is responsible for drawing into the visible area.
- :focus-callback**  
A function called when the pane gets or loses the input focus, or `nil`. The default is `nil`.
- :resize-callback**  
A function called when the pane is resized, or `nil`. The default is `nil`.
- :create-callback**  
A function called just after the pane is created.
- :destroy-callback**  
A function called just before the pane is destroyed.
- :graphics-options**  
A platform-specific plist of options controlling how graphics are drawn.

`:draw-with-buffer`

A boolean controlling whether output is buffered, on Windows and Motif.

#### 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`).

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 User Guide and Reference Manual* for a description of this and other functions for manipulating Gesture Spec objects.

**Note:** on Cocoa you cannot receive `command` key gestures via Gesture Spec mapping in *input-model*. To receive `command` key gestures you should add corresponding menu items with accelerators. See `menu-item` for information about accelerators.

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`, `:meta` and `:hyper`. The `:meta` modifier will be the `Alt` key on most keyboards. On Cocoa, the `:hyper` modifier is interpreted as the `command` key for button and motion gestures. On Windows, the `:hyper` modifier is currently never

generated, so gestures mappings using it will never be invoked. `:third-press` and `:nth-press` are supported only on Cocoa and Motif.

Key mappings are intended for detecting low-level keyboard input. In a key mapping, *gesture* should be a list

```
(:key [keyname] action [modifiers] *)
```

where the optional *keyname* is a character naming a key (no modifiers) or one of the valid Gesture Spec keywords, *action* is one of `:press` or `:release` and each modifier is one of the keywords `:shift`, `:control` and `:meta`. The callback will receive a Gesture Spec object, with its data set to an integer ASCII code or a keyword representing the primary item on the key and its modifiers representing the set of modifiers pressed. The `:meta` modifier will be the `ALT` key on most keyboards. On Cocoa, the `:hyper` modifier is interpreted as the `command` key for `:key` input.

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:

`:glyph`            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.

When *draw-with-buffer* is true, display of the `output-pane` (that is drawing the background and calling the *display-callback*) is done by first drawing to a pixmap buffer, and then drawing from that buffer. This is useful to avoid flickering if the display is complex. The default value of *draw-with-buffer* is `nil`.

- Notes
1. *draw-with-buffer* is typically useful for a `pinboard-layout` with large number of pinboard objects, or any other feature that may cause it to flicker.
  2. The GTK+ and Cocoa libraries always buffer, so *draw-with-buffer* is ignored on these platforms.

Example            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 &amp;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-nil, 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 &amp;key &amp;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.

**Notes** Only a keyword value for *adjust* should be supplied when *pane* is a `column-layout` or `row-layout`.

**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-descendant-child-with-focus***Function*

Summary	Finds the child with the input focus.	
Signature	<code>pane-descendant-child-with-focus <i>pane</i> =&gt; <i>result</i></code>	
Arguments	<i>pane</i>	A pane or layout.
Values	<i>result</i>	A pane or <code>nil</code> .
Description	<p>The function <code>pane-descendant-child-with-focus</code> attempts to find the pane inside <i>pane</i> that currently has the input focus, and returns this pane if successful.</p> <p><code>pane-descendant-child-with-focus</code> may return <code>nil</code> if it does not find a pane with the focus.</p>	
See also	<code>pane-has-focus-p</code>	

**pane-got-focus***Generic Function*

Summary	A function called when the focus is set programmatically.	
Package	<code>capi</code>	
Signature	<code>pane-got-focus <i>interface pane</i></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</p>	

which is hidden inside a `tab-layout` or `switchable-layout`. Your method can check for this case and modify the layout as required.

See also `set-object-automatic-resize`

## pane-has-focus-p

*Generic Function*

Summary Determines whether a pane has the focus.

Package `capi`

Signature `pane-has-focus-p pane => focusp`

Arguments `pane` A CAPI element.

Values `focusp` A boolean.

Description The function `pane-has-focus-p` is the predicate for whether `pane` currently has the input focus.

**Note:** On Motif, `pane-has-focus-p` cannot be used in menu functions such as the *enabled-function* or *popup-callback* of a menu item. It will always return `nil`, because the focus is on the menu button when the user clicks on it.

See also `accepts-focus-p`  
`pane-descendant-child-with-focus`  
`set-object-automatic-resize`

## pane-initial-focus

*Generic Function*

Summary Gets or sets the initial focus pane.

Package `capi`

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	<p>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.</p> <p><code>(setf pane-initial-focus)</code> may be used to set the initial focus pane, but only before <i>pane-with-children</i> has been created. If the setter is called after <i>pane-with-children</i> has been created, an error is signalled.</p> <p><i>pane-with-children</i> should be a pane with child panes such as a layout, an interface, a button-panel or a toolbar.</p>
See also	<code>pane-has-focus-p</code>

pane-interface-copy-object  
pane-interface-copy-p  
pane-interface-cut-object  
pane-interface-cut-p  
pane-interface-deselect-all  
pane-interface-deselect-all-p  
pane-interface-paste-object  
pane-interface-paste-p  
pane-interface-select-all  
pane-interface-select-all-p  
pane-interface-undo  
pane-interface-undo-p

### *Generic Functions*

Summary      Implements "edit/select operations" and the associated predicates for the active pane.

Signature      `pane-interface-copy-object` *pane interface*  
`pane-interface-copy-p` *pane interface*  
`pane-interface-cut-object` *pane interface*  
`pane-interface-cut-p` *pane interface*  
`pane-interface-deselect-all` *pane interface*  
`pane-interface-deselect-all-p` *pane interface*  
`pane-interface-paste-object` *pane interface*  
`pane-interface-paste-p` *pane interface*  
`pane-interface-select-all` *pane interface*  
`pane-interface-select-all-p` *pane interface*  
`pane-interface-undo` *pane interface*  
`pane-interface-undo-p` *pane interface*

Description	<p>The active pane "edit/select operations" call these generic functions when the active pane does not specify how to perform the operation. Do not call these directly.</p> <p><i>interface</i> is the top level interface of the pane. The predicate functions (those with names ending with <code>-p</code>) should return true if the operation can be performed. The other functions should perform the operations.</p> <p>You can implement your own methods specializing on pane and interface classes.</p>
See also	<code>active-pane-copy</code>

## 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 <i>pane interface</i> =&gt; <i>items</i></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>	

To specify items for menus associated with panes in your interfaces, define `pane-popup-menu-items` methods specialized on your interface class.

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`.

#### Notes

1. `pane-popup-menu-items` is not supported for text panes on Cocoa such as `rich-text-pane`.
2. `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.
3. 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 `edgraph`:

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>	

## pane-supports-menus-with-images

*Function*

Summary	Tests whether a pane supports menus with images.	
Signature	<code>pane-supports-menus-with-images</code> <i>pane</i> => <i>result</i>	
Arguments	<i>pane</i>	A displayed CAPI pane.
Values	<i>result</i>	A boolean.
Description	<p>The function <code>pane-supports-menus-with-images</code> returns <code>t</code> if the pane supports menus with images. This means that the menus display both the images and the text correctly.</p> <p>See the <i>image-function</i> of <code>menu</code> for details of creating a menu with images.</p> <p>When <code>pane-supports-menus-with-images</code> returns <code>nil</code>, menus can display images, but not together with text at the same item They may also display images with transparency incorrectly.</p> <p>Whether the pane supports menus with images depends on the library in which it is displayed. Support is currently limited to GTK+ and Cocoa.</p>	
See also	<code>menu</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	<code>capi</code>
Superclasses	<code>text-input-pane</code>
Initargs	<code>:overwrite-character</code> A <code>base-char</code> .
Readers	<code>password-pane-overwrite-character</code>

**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 `#\*`.

**Example**

```
(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	<code>play-sound</code> <i>sound</i> &key <i>wait</i>
Arguments	<i>sound</i> A sound object returned by <code>load-sound</code> . <i>wait</i> A generalized boolean.
Description	The function <code>play-sound</code> plays the loaded sound <i>sound</i> . If <i>wait</i> is true then <code>play-sound</code> will not return until <i>sound</i> has finished playing. That is, it plays the sound synchronously. The default value of <i>wait</i> is <code>nil</code> . <b>Note:</b> <code>:wait t</code> is only implemented on Microsoft Windows.
See also	<code>load-sound</code> <code>stop-sound</code>

## 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 subclass of <code>static-layout</code> and so it 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>static-layout</code>
Subclasses	<code>simple-pinboard-layout</code>
Initargs	<code>:highlight-style</code> A keyword.

## Description

When a `pinboard-layout` lays out its children, it positions them at the  $x$  and  $y$  specified as hints (using `:x` and `:y`), and sizes them to their minimum size (which can be specified using `:visible-min-width` and `:visible-max-width`).

By default, the `pinboard-layout` is made sufficiently large to accommodate all of its children, as specified by `fit-size-to-children` in the superclass `static-layout`. If you need the sizing capabilities, then use the class `simple-pinboard-layout` which surrounds a single child, and adopts the size constraints of that child.

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 function `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`.

**Notes** If redrawing flickers on Microsoft Windows or Motif, perhaps because there are many pinboard objects, you can pass the `output-pane` initarg `:draw-with-buffer t`, which uses a pixmap to buffer the output before drawing it to the screen. See `output-pane` for more information.

**Example** 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/Setup.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`
- `static-layout`

## pinboard-object

*Class*

Summary	Provides a rectangular area in a <code>pinboard-layout</code> 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. The following initargs are geometry hints, influencing the initial size and position of a <code>pinboard-object</code> : <code>:x</code> The <code>x</code> position of the pinboard object in the pinboard. <code>:y</code> The <code>y</code> 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.
<code>:external-max-height</code>	The maximum height of the pinboard object in the pinboard.
<code>:visible-min-width</code>	The minimum visible width of the pinboard object.
<code>:visible-min-height</code>	The minimum visible height of the pinboard object.
<code>:visible-max-width</code>	The maximum visible width of the pinboard object.
<code>:visible-max-height</code>	The maximum height of the pinboard object.
<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 `pinboard-object-pinboard`  
`pinboard-object-activep`  
`pinboard-object-graphics-args`

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 themselves 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).

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`.

Example 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 *x* and *y*. It determines this by mapping over every pinboard object within the pinboard until it finds one for which the generic function `over-pinboard-object-p` returns *t*.

**Example**

```
(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)
```

**See also**

- `over-pinboard-object-p`
- `pinboard-object-overlap-p`
- `pinboard-object`
- `pinboard-layout`

## pinboard-object-graphics-arg

## *Generic Function*

**Summary** Gets or sets the value of a particular drawing option in a `pinboard-object`.

**Package** `capi`

**Signature** `pinboard-object-graphics-arg` *self* *keyword* => *value*

**Signature** `(setf pinboard-object-graphics-arg)` *value*  
*self* *keyword* => *value*

**Arguments** *self* A `pinboard-object`.

	<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-graphics-arg</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		Tests whether a specified region overlaps with the region of a pinboard object.
Package	<code>capi</code>	
Signature		<code>pinboard-object-overlap-p</code> <i>pinboard-object</i> <i>top-left-x</i> <i>top-left-y</i> <i>bottom-right-x</i> <i>bottom-right-y</i> => <i>result</i>
Description		The generic function <code>pinboard-object-overlap-p</code> returns true if the region of the pinboard object <i>pinboard-object</i> overlaps with the region specified by the other arguments.
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</code> <i>self</i> => <i>x</i> , <i>y</i>	
Signature	<code>(setf pinboard-pane-position)</code> <i>x</i> , <i>y</i> <i>self</i> => <i>x</i> , <i>y</i>	
Arguments	<i>self</i>	A <code>pinboard-object</code> or <code>simple-pane</code> .
Values	<i>x</i> , <i>y</i>	The horizontal and vertical coordinates in the <code>pinboard-layout</code> parent of <i>self</i> .
Description	The generic function <code>pinboard-pane-position</code> returns as multiple values <i>x</i> , <i>y</i> the coordinates of <i>self</i> inside its parent <code>pinboard-layout</code> .  <code>(setf pinboard-pane-position)</code> sets the location of <i>self</i> in its parent.	

```

Example      (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 callback-error-handler => result, successp`

Arguments `pane` A CAPI pane or interface.

<i>message</i>	A string or <code>nil</code> .
<i>modal, screen, focus, owner, x, y, and position-relative-to</i>	These are passed to <code>display-dialog</code> .
<i>title</i>	A string specifying the title of the dialog window.
<i>title-font</i>	The font used in the title.
<i>value-function</i>	Controls the value returned, and whether a value can be returned.
<i>exit-function</i>	Called on exiting the dialog.
<i>apply-function, apply-check, apply-button</i>	Define the callback, check function and title of an <b>Apply</b> button.
<i>ok-function, ok-check, ok-button</i>	Define the callback, check function and title of an <b>OK</b> button.
<i>no-button, no-function</i>	Define the title and callback of a <b>No</b> button.
<i>all-button, all-function</i>	Define the title and callback of an <b>All</b> button.
<i>cancel-button</i>	Defines the title of a <b>Cancel</b> button.
<i>help-button, help-function</i>	Define the title and callback of a <b>Help</b> 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> .

*buttons-uniform-size-p*

Controls relative button sizes.

*foreground, background*

Specify colors.

*font*

A font or a font description.

*button-font*

A font or a font description.

*button-container* A layout controlling where the buttons of the dialog appear.

*continuation*

A function or `nil`.

*callback-error-handler*

A function designator or `nil`.

Values

*result*

The result of *value-function*, or *pane*, or `nil`.

*successp*

`nil` if the dialog was cancelled, `t` otherwise.

Description

The function `popup-confirmer` 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.

Generally the `Return` key selects the dialog's **OK** button and the `Escape` key selects the **Cancel** button, if there is one.

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 true 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.

*callback-error-handler*, if non-`nil`, should be a function designator for a function of one argument which is a condition, like the *handler-function* in `c1:handler-bind`. The handler is established (by `c1:handler-bind` with type `c1:error`) around each callback call inside the scope of `popup-con-`

`firmer` or `display-dialog`. In recursive calls, only the handler of the innermost call to `popup-confirmer` or `display-dialog` is established.

`callback-error-handler` can use `current-popup` to find the popup (first argument to the innermost call of `display-dialog` or `popup-confirmer`).

If `callback-error-handler` wants to do a non-local exit, it should either call `abort-callback` to abort the callback but leave the dialog, or `exit-dialog` (or `abort-dialog`) to exit (or abort) the dialog.

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*.

## Notes

1. On Windows and Motif, the effect of `callback-error-handler` can be achieved by using `cl:handler-bind` around the call to `display-dialog` or `popup-confirmer` (the handler will also handle errors during raising the dialog, but these are not expected to happen). On Cocoa, using such an error handler does not necessarily work, because the callback may happen in another process. `callback-error-handler` ensures that the callback is in the scope of the handler on all platforms. From the same reason the handler should not rely on the dynamic environment (including catchers and restarts), and needs to use `current-popup` to find its "context" and use `abort-callback`, `exit-dialog` or `abort-dialog` for non-local exit.
2. If the callback itself calls `popup-confirmer` or `display-dialog`, the `callback-error-handler` handler will stay until the callback returns. Unless the recursive call handles the error, the handler of the outer call may be called to handle it, and needs to be written to deal with this possibility correctly. If the handler inside a recursive call needs to

access the popup that was used in the same call that the handler was used, it should close over it, because `current-popup` returns the innermost one.

3. A handler that is established by the callback (by `cl:handler-bind` or `cl:handler-case`) is inside the scope of the *callback-error-handler*, and therefore will be called first.

## Example

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)
```

This example illustrates the use of *callback-error-handler*.

```

(defun my-error-handler (condition)
  (let ((pane (capi:current-popup)))
    (capi:display-message
     "Error inside dialog: ~a : ~a"
     (capi:capi-object-name pane)
     condition)
    (capi:abort-callback)))

(let*
  ((foo-callback
    (lambda ()
      (let ((md (make-instance
                 'capi:push-button
                 :text "Error inside Callback-Error-
Handler"
                 :name "Chicken"
                 :callback-type :data
                 :data "Twisted ankle."
                 :callback 'error)))
        (capi:popup-confirmer
         md nil
         :callback-error-handler 'my-error-handler))))
   (foo (make-instance
        'capi:push-button
        :text
        "Popup confirmer with Callback-Error-Handler"
        :callback-type :none
        :callback foo-callback))
   (bar (make-instance
        'capi:push-button
        :text "Error without a handler"
        :callback-type :data
        :data "Broken leg."
        :callback 'error)))
  (capi:contain (list foo bar)))

```

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	<code>capi</code>
Superclasses	<code>item</code>
Initargs	<p><code>:menu</code>                   A menu or <code>nil</code>.</p> <p><code>:menu-function</code></p> <p>                              A function designator or <code>nil</code>.</p>
Accessors	<p><code>popup-menu-button-menu</code></p> <p><code>popup-menu-button-menu-function</code></p>
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</code>	<i>button</i> => <i>text</i>
Arguments	<i>button</i>	A <code>button</code> .
Values	<i>text</i>	A string.
Description	<p>The generic function <code>print-capi-button</code> is used to generate the text for a <code>button</code>.</p> <p>You can add methods for your own <code>button</code> 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	<i>item</i>	An <code>item</code> or an Lisp object.
	<i>collection</i>	A <code>collection</code> or any Lisp object.
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 <code>collection</code> prints using the first of these which returns non-nil: 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 <code>collection</code> 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

```
(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)
```

In this example we provide our own `print-collection-item` method:

```
(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** `capi`

**Signature** `print-editor-buffer` *buffer* &key *start end printer interactive font*

**Description** The `print-editor-buffer` function prints the contents of *buffer* to *printer*, which is the current printer by default.

By default the entire editor buffer is printed, but by specifying *start* and *end* to be editor points, a part of the buffer can be printed. See the *LispWorks Editor User Guide* for information about editor points.

If *interactive* is `t`, the default value, then a printer dialog is displayed.

*font* is interpreted as described for `print-text`.

See also `print-file`  
`print-text`

**print-file***Function*

**Summary** Prints the contents of a specified file.

**Package** `capi`

**Signature** `print-file` *file* &key *printer interactive font*

Description	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.  <i>font</i> is interpreted as described for <code>print-text</code> .
See also	<code>print-editor-buffer</code> <code>print-text</code>

## 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	The function <code>print-rich-text-pane</code> prints the contents in <i>pane</i> .  <i>jobname</i> is the name of the print job. The default value is <code>nil</code> , meaning that the name "Document" is used.  <i>printer</i> is the printer to use. The default value is <code>nil</code> , meaning that the <code>current-printer</code> is used.	

*interactive*, if true, specifies that a `print-dialog` is displayed before printing. The default value of *interactive* is `t`.

*selection* is a boolean specifying what to print. If true, only the current selection is printed. If `nil`, all the contents of *pane* are printed. The default value is `nil`.

**Note:** `print-rich-text-pane` is supported only on Microsoft Windows.

See also `rich-text-pane`

## print-text

*Function*

Summary Prints plain text to a printer.

Package `capi`

Signature `print-text line-function &key printer tab-spacing interactive font`

Description The `print-text` function prints plain text to a printer specified by *printer*, and defaulting to the current printer.

The *line-function* is called repeatedly with no arguments to enumerate the lines of text. It should return `nil` when the text is exhausted.

The *tab-spacing* argument, which defaults to 8, specifies the number of spaces printed when a tab character is encountered.

If *interactive* is `t`, then a print dialog is displayed. This is the default behavior.

*font* should be a `gp:font` object, or a Font Description object, or a symbol which is a font alias as defined by `define-font-alias`. 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 *font* is a Font Description for a fixed pitch font of size 10.

See also `print-editor-buffer`  
`print-file`

## printer-configuration-dialog

*Function*

Summary Displays a dialog allowing the user to configure printers.

Package `capi`

Signature `printer-configuration-dialog &key screen owner`

Description The `printer-configuration-dialog` function displays the printer configuration dialog that allows users to add and configure PostScript printers.

This applies only on GTK+ and Motif.

The *screen* argument specifies a CAPI screen on which to display the dialog. The *owner* 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.

The general options that are available are described under `install-postscript-printer`. In addition, printer-specific options (which are defined in the printer PPD file) are available.

The printers that are visible in the dialog are defined by files in the directories in the list `*printer-search-path*`.

See also `install-postscript-printer`  
`*printer-search-path*`

**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 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.</p> <p><code>printer-metrics-dpi-x</code> and <code>printer-metrics-dpi-y</code> 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.</p> <p><code>printer-metrics-height</code> and <code>printer-metrics-width</code> respectively return the height and width of the printable area in millimeters.</p> <p><code>printer-metrics-left-margin</code> and <code>printer-metrics-top-margin</code> respectively return the current left margin and current top margin of the printable area in millimeters.</p> <p><code>printer-metrics-max-height</code> and <code>printer-metrics-max-width</code> respectively return the greatest possible height and width of the printable area in millimeters.</p>

`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 GTK+ and Motif.

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	<code>capi</code>	
Signature	<code>printer-port-handle</code>	&optional <i>port</i> => <i>handle</i>
Arguments	<i>port</i>	A printer port.
Values	<i>handle</i>	Platform-dependent.
Description	<p>The function <code>printer-port-handle</code> returns a platform-dependent value which represents the underlying handle to the printer port.</p> <p>On Microsoft Windows, <i>handle</i> is the HDC for the printer device.</p> <p>If <i>port</i> is passed it should be the value bound to <i>var</i> in <code>with-print-job</code>. If <i>port</i> is not supplied it defaults to the current printer port (dynamically bound within <code>with-print-job</code>).</p>	
See also	<code>with-print-job</code>	

## printer-port-supports-p

*Function*

Summary	Detects if the printer port can support a certain feature.	
Package	<code>capi</code>	
Signature	<code>printer-port-supports-p</code>	<i>feature</i> &optional <i>port</i> => <i>supportedp</i> , <i>validp</i>
Arguments	<i>feature</i>	A keyword.
	<i>port</i>	A printer port.
Values	<i>supportedp</i>	A boolean.
	<i>validp</i>	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         `capi`

Initial value    `("~/lispworks-printers/" nil)`

Description     The variable `*printer-search-path*` specifies where to look for printer definition files.

                  This applies only on GTK+ and Motif.

                  The value is a list containing directory pathname designators specifying where to look for printer definition files. The list can also include the value `nil`, which is interpreted as the `printers` directory in the LispWorks library.

                  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.

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.

The first path in the `*printer-search-path*` 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 `*printer-search-path*` and then tries to save it, the system prompts for whether to overwrite the original or save it in the "local" directory.

The printer files can be copied to other directories, on the same machine, and hence to install printers in different directories.

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.

## process-pending-messages

*Function*

Summary	Processes all the pending messages in the current process.
Package	<code>capi</code>
Signature	<code>process-pending-messages</code> <i>ignored</i> => <code>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 &amp;key color colors owner =&gt; result, successp</code>								
Arguments	<table><tr><td><i>message</i></td><td>A string.</td></tr><tr><td><i>color</i></td><td>A color specification.</td></tr><tr><td><i>colors</i></td><td>A list.</td></tr><tr><td><i>owner</i></td><td>An owner window.</td></tr></table>	<i>message</i>	A string.	<i>color</i>	A color specification.	<i>colors</i>	A list.	<i>owner</i>	An owner window.
<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 GTK+ and 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 <b>Yes</b> and <b>No</b> buttons.	
Package	<code>capi</code>	
Signature	<code>prompt-for-confirmation message &amp;key screen owner cancel-button default-button continuation =&gt; 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 <b>Yes</b> and <b>No</b> buttons. When either <b>Yes</b> or <b>No</b> is pressed, it returns two values:</p> <ul style="list-style-type: none"> <li>• a boolean indicating whether <b>Yes</b> was pressed</li> <li>• <code>t</code> (for compatibility with other prompt functions)</li> </ul> <p><i>cancel-button</i> specifies whether a <b>Cancel</b> button also appears on the dialog. When <b>Cancel</b> 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 <b>Yes</b>, the value <code>:cancel</code> means <b>Cancel</b>, and any other value means <b>No</b>. The default value of <i>default-button</i> 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>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-confirmation</code>. On Cocoa, passing <i>continuation</i> causes the dialog to be made as a window-modal sheet and <code>prompt-for-confirmation</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>	
Example	<pre>(capi:prompt-for-confirmation "Continue?")</pre>	

```
(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	<code>capi</code>	
Signature	<code>prompt-for-directory message &amp;key if-does-not-exist pathname file-package-is-directory pane-args popup-args owner continuation =&gt; result, successp</code>	
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>file-package-is-directory</i>	A generalized boolean.
	<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.
Values	<i>continuation</i>	A function or <code>nil</code> .
	<i>result</i>	A directory pathname, or <code>nil</code> .
	<i>successp</i>	A boolean.

Description     The function `prompt-for-directory` prompts the user for a directory pathname using a dialog box. Like all the `prompt-ers`, `prompt-for-directory` returns two values: the directory pathname and a flag indicating success. The *successp* flag will be `nil` if the dialog was cancelled, and `t` otherwise.

On Windows and Motif, if *if-does-not-exist* is `:ok`, a non-existent directory can be chosen. When set to `:prompt`, if a non-existent directory is chosen, the user is prompted for whether the directory should be created. When set to `:error`, the user cannot choose a non-existent directory. The default value of *if-does-not-exist* is `:prompt`.

On Cocoa it is never possible to choose a non-existent directory, and the value of *if-does-not-exist* is ignored.

*pathname*, if non-`nil`, supplies an initial directory for the dialog. The default value for *pathname* is `nil`, and with this value the dialog initializes with the current working directory.

*file-package-is-directory* is handled as by `prompt-for-file`.

*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-directory`. On Cocoa, passing *continuation* causes the dialog to be made as a window-modal sheet and `prompt-for-directory` returns immediately, leaving 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** `capi`

**Signature** `prompt-for-file message &key pathname ok-check filter filters if-exists if-does-not-exist file-package-is-directory operation owner pane-args popup-args continuation => filename, successp, filter-name`

**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>file-package-is-directory</i>	A generalized boolean.
<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 `prompt-for-file` prompts the user for a file using a dialog box.

*pathname*, if non-nil, is a pathname designator providing a default filename for the dialog.

*ok-check*, if non-nil, should be a function which takes a pathname designator argument and returns a true value if the pathname is valid.

*filter* specifies the initial filter expression. The default value is `*.*`. An example filter expression with multiple filters is `*.LISP;*.LSP`.

*filter* is used on all platforms. However on Motif, if *filter* contains multiple file types, only the first of these is used.

On Cocoa `prompt-for-file` supports the selection of application bundles as files if they match the filter. For example, they will match if the filter expression contains `*.app` or `*.*`.

*filters* is a property list of filter names and filter expressions, presenting filters which the user can select in the dialog. If the *filter* argument is not one of the expressions in *filters*, an extra filter called `Files` is added for this expression.

On Microsoft Windows the default value of *filters* is:

```
("Lisp Source Files" "*.LISP;*.LSP"
 "Lisp Fasls" "*.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.

*file-package-is-directory* controls how to treat file packages on Cocoa. By default it is `nil`, which means that a file package is treated as file. If *file-package-is-directory* is non-`nil`, the a file package is treated as a directory. *file-package-is-directory* corresponds to the `treatsFilePackagesAsDirectories` method of `NSSavePanel` in Cocoa. It has no effect on other platforms.

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

`capi`

Signature

```
prompt-for-files message &key pathname ok-check filter filters
if-exists if-does-not-exist file-package-is-directory operation owner
pane-args popup-args continuation => filenames, successp, filter-name
```

Values

```
filenames      A list.
successp       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**      `capi`

Signature	<code>prompt-for-font</code> <i>message</i> &key <i>font</i> <i>owner</i> => <i>result</i> , <i>successp</i>	
Arguments	<i>message</i>	A string.
	<i>font</i>	A font, a font description, or <code>nil</code> .
	<i>owner</i>	An owner window, or <code>nil</code> .
Values	<i>result</i>	A font, or <code>nil</code> .
	<i>successp</i>	A boolean.
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</code> <i>message</i> &key <i>package</i> <i>initial-value</i> <i>evaluate</i> <i>quotify</i> <i>ok-check</i> <i>value-function</i> <i>pane-args</i> <i>popup-args</i> <i>continuation</i> => <i>result</i> , <i>okp</i>	

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 <code>*package*</code> if not. If <i>evaluate</i> is non-nil 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-nil, 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 &amp;key package initial-value value-function pane-args popup-args continuation =&gt; 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-nil 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-nil, 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.

Example

```
(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 &amp;key pane-args popup-args interaction choice-class continuation =&gt; 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 &amp;key min max initial-value ok-check pane-args popup-args continuation =&gt; 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> .

<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-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.

Example

```
(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.

#### Example

```
(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 &amp;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-nil, 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</code> <i>items message</i> &key <i>choice-class interaction value-function pane-args popup-args continuation</i> => <i>result, successp</i>	
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*.

### Example

```
(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*:

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.

Example `(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, Cocoa and GTK+ 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, GTK+ 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 and GTK+, 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.

Example

```
(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> &amp;key <i>force</i> =&gt; <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><b>Note:</b> <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      A class supporting progress-bar and slider.

Package      capi

Superclasses  None

Subclasses   progress-bar
              scroll-bar
              slider

Initargs     :start          The lowest value of the range.
              :end            The highest value of the range.
              :slug-start    The start of the slug, corresponding to the
                              current value of the range.
              :slug-end      The end of the slug.
              :callback      Called when the user changes the value.
              :orientation  One of :horizontal (the default) or
                              :vertical.

```

Accessors	<code>range-start</code> <code>range-end</code> <code>range-slug-start</code> <code>range-slug-end</code> <code>range-callback</code> <code>range-orientation</code>
Description	The class <code>range-pane</code> exists to support the <code>progress-bar</code> and <code>slider</code> classes. Consult the reference pages for <code>progress-bar</code> and <code>slider</code> for further information.
See also	<code>progress-bar</code> <code>slider</code>

## range-set-sizes

## Function

Summary	Set values in a <code>range-pane</code> .										
Signature	<code>range-set-sizes range-pane &amp;key start end slug-start slug-end redisplay</code>										
Arguments	<table> <tr> <td><code>range-pane</code></td> <td>A <code>range-pane</code>.</td> </tr> <tr> <td><code>start</code></td> <td>A real number or nil.</td> </tr> <tr> <td><code>end</code></td> <td>A real number or nil.</td> </tr> <tr> <td><code>slug-start</code></td> <td>A real number or nil.</td> </tr> <tr> <td><code>slug-end</code></td> <td>A real number or nil.</td> </tr> </table>	<code>range-pane</code>	A <code>range-pane</code> .	<code>start</code>	A real number or nil.	<code>end</code>	A real number or nil.	<code>slug-start</code>	A real number or nil.	<code>slug-end</code>	A real number or nil.
<code>range-pane</code>	A <code>range-pane</code> .										
<code>start</code>	A real number or nil.										
<code>end</code>	A real number or nil.										
<code>slug-start</code>	A real number or nil.										
<code>slug-end</code>	A real number or nil.										
Description	<p>The function <code>range-set-sizes</code> set the values in the <code>range-pane</code> <code>range-pane</code> for any value of <code>start</code>, <code>end</code>, <code>slug-start</code> or <code>slug-end</code> that is supplied as non-nil.</p> <p>For each of <code>start</code>, <code>end</code>, <code>slug-start</code> and <code>slug-end</code>, if the value is <code>nil</code> or not supplied, the corresponding value in <code>range-pane</code> is not changed.</p> <p>If <code>redisplay</code> is true then <code>range-pane</code> is redisplayed with the new values.</p>										

The default value of *redisplay* is `t`.

Notes The values can be also set individually by the accessors (`setf range-start`) and so on. `range-set-sizes` has the advantage over the accessors that it causes fewer calls to `redisplay`.

See also `range-pane`

## 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> redisplays <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	<p>The function <code>redraw-pinboard-layout</code> causes any pinboard objects within the given rectangle of the pinboard layout to get redrawn.</p> <p>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>.</p>
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	<p>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>.</p>
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

```
capi
```

Signature

```
remove-items collection list-or-predicate
```

Arguments

```
collection      A collection.
list-or-predicate A list, or a function of one argument
                    returning a boolean value.
```

Description

The generic function `remove-items` removes from the `collection` *collection* those items determined by *list-or-predicate*.

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	<p>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.</p> <p><i>interface</i> is a CAPI interface to be displayed as a dialog.</p> <p>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>.</p>	

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** `capi`

**Signature** `report-active-component-failure` *pane* *component-name* *error-string* *function-name* *hresult*

**Arguments**

- pane* An `ole-control-pane`.
- component-name* A string or `nil`.
- error-string* A string.
- hresult* An integer or `nil`.

**Description** The generic function `report-active-component-failure` is used to report on failures to find or create a component. *component-name* is the name of the component it tried to find. *error-string* is the error string. *function-name* is the name of the function that actually failed. *hresult* is the `hresult` that came back. It may be `nil` if the error is that the `guid` 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  
`examples/com/ole/simple-container/doc-viewer-pair.lisp`

See also      `ole-control-pane`

## reuse-interfaces-p

*Function*

Summary      Determines whether global interface re-use is enabled.

Package      `capi`

Signature      `reuse-interfaces-p => result`

Signature      `(setf reuse-interfaces-p) value => value`

Arguments      *value*              A boolean.

Values         *result*              A boolean.

Description      The function `reuse-interfaces-p` is the predicate for whether global interface re-use is enabled.

The function `(setf reuse-interfaces-p)` 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</code> <i>pane</i> &key <i>selection</i> => <i>result</i>	
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> &amp;rest <i>args</i> =&gt; <i>result</i>, <i>result?</i></code>	
Arguments	<i>pane</i>	A <code>rich-text-pane</code> .
	<i>operation</i>	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	<p>The valid values of <i>operation</i> on Microsoft Windows and Cocoa are:</p> <p><code>:pasted</code>, <code>:cutp</code> OR <code>:copyp</code></p> <p><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.</p> <p><code>:paste</code>, <code>:cut</code>, OR <code>:copy</code></p> <p>Performs the indicated operation.</p> <p><code>:select-all</code>    Selects all the text.</p> <p><code>:set-selection</code></p> <p><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).</p> <p><code>:get-selection</code></p> <p>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.</p> <p><code>:can-undo</code> OR <code>:can-redo</code></p> <p><i>result</i> is a boolean indicating whether it is currently possible to perform an <code>:undo</code> or <code>:redo</code> operation.</p> <p><code>:undo</code></p> <p>Undoes the last editing operation. Note that, after typing, it is the whole input, rather than a single character, that is undone. The</p>	

`: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.

Example

```
(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 =&gt; 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 keywords <code>:divider</code> and <code>:separator</code> which automatically create a divider or separator as a child of the <code>row-layout</code>. The user can move a divider, but cannot move a separator.</p> <p>When specifying <code>:ratios</code> in a row with <code>:divider</code> or <code>:separator</code>, you should use <code>nil</code> to specify that the divider or separator is given its minimum size.</p>	
Compatibility note	<p><code>*layout-divider-default-size*</code> and <code>row-layout-divider</code> are not supported in LispWorks 4.4 and later.</p>	

## Example

```
(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))
```

See also `column-layout`

## screen

*Class*

**Summary** A `screen` is an object that represents the known monitor screens.

**Package** `capi`

Superclasses	<code>capi-object</code>	
Subclasses	<code>color-screen</code> <code>mono-screen</code>	
Initargs	<code>:width</code>	The width in pixels of the screen.
	<code>:height</code>	The height in pixels of the screen.
	<code>:number</code>	The screen number.
	<code>:depth</code>	The number of color planes in the screen.
	<code>:interfaces</code>	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	<pre>(setq screen (capi:convert-to-screen)) (capi:screen-width screen)</pre>	

```
(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	<code>capi</code>
Signature	<code>screen-active-interface <i>screen</i> =&gt; <i>interface</i></code>
Arguments	<i>screen</i> A screen or document-container
Values	<i>interface</i> An interface, or nil.
Description	The function <code>screen-active-interface</code> returns the currently active interface on the screen <i>screen</i> , or nil if no CAPI interface is active or if this cannot be determined.  <code>screen-active-interface</code> also works with <code>document-container</code> , returning the active interface within the container.
See also	<code>document-container</code> <code>screen</code>

## screen-active-p

*Function*

Summary	Determines whether a screen is active.
Package	<code>capi</code>

Signature	<code>screen-active-p</code>	<code>screen =&gt; 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</code>	<code>screen =&gt; 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 `screen-internal-geometry screen => x, y, width, height`

Arguments `screen` A screen.

Values `x` An integer.

`y` An integer.

`width` An integer.

`height` An integer.

Description The function `screen-internal-geometry` returns the geometry (as `x`, `y`, `width` and `height`) 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.

On Microsoft Windows `screen-internal-geometry` works with `document-container`, returning the current size of the container (which may vary over time).

See also `document-container`  
`screen`

## screens

## Function

Summary Returns the active screens for a library.

Package `capi`

Signature `screens &optional library => result`

Arguments `library` A library name, a list, or `:any`.

Values `result` A list.

Description The function `screens` returns as a list all the active screens for `library`.

A library name is a keyword naming a library, currently `:win32` on Microsoft Windows, `:gtk` on GTK+, `:motif` on Motif and `:cocoa` on Mac OS X with the native GUI.

*library* can be a library name, or a list of library names, or the keyword `:any`, meaning all the libraries. The default value of *library* is the result of `default-library`.

See also `default-library`  
`screen`

## scroll

## Generic Function

Summary Moves the scrollbar and calls the *scroll-callback*.

Package `capi`

Signature `scroll self scroll-dimension scroll-operation scroll-value &rest options`

Arguments *self* A pane that supports scrolling.  
*scroll-dimension* `:vertical`, `:horizontal` or `:pan`.  
*scroll-operation* `:move`, `:step` or `:page`.  
*scroll-value* An integer, or a list of two integers, or a keyword, or a list of two keywords.  
*options* A list.

Description The generic function `scroll` works for panes that support scrolling - these are subclasses of `output-pane` and `layout`.  
`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> <p><i>callback</i> can be <code>nil</code>, 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:</p>	

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`

## scroll-if-not-visible-p

*Generic Function*

Summary `Accesses the scroll-if-not-visible-p attribute of a pane.`

Signature `scroll-if-not-visible-p pane => value`  
`(setf scroll-if-not-visible-p) value pane`

Values	<i>value</i>	One of <code>t</code> , <code>nil</code> or <code>:non-mouse</code> .
Method Signature	<code>scroll-if-not-visible-p</code>	<code>simple-pane</code> <code>(setf scroll-if-not-visible-p) <i>value</i> simple-pane</code>
Description	<p>The generic function <code>scroll-if-not-visible-p</code> accesses the <i>scroll-if-not-visible-p</i> attribute of a pane.</p> <p>The value of this attribute has these meanings:</p> <p><code>t</code> When <i>pane</i> is given the input focus, and it is not fully visible, and its parent can be scrolled to make the pane visible, then the parent is scrolled automatically. This is the default value.</p> <p><code>nil</code> Never scroll the parent to make a pane visible.</p> <p><code>:non-mouse</code> Like <code>t</code>, except that it does not scroll when the focus is given as a result of a mouse click in <i>pane</i>.</p> <p><code>scroll-if-not-visible-p</code> is called by CAPI each time it may need to scroll the parent. The method on <code>simple-pane</code> returns a value that is kept internally, and can be set by the default <code>setf</code> method.</p> <p>You can specialize <code>scroll-if-not-visible-p</code> on your classes, but note that it is called often when the user clicks on any pane, so it must be reasonably fast.</p> <p>The setter sets the <i>scroll-if-not-visible-p</i> attribute. It is called when the <code>initarg :scroll-if-not-visible-p</code> is used in making a <code>simple-pane</code> (or a subclass) instance, and can be called by your program. <i>value</i> must be <code>t</code>, <code>nil</code> or <code>:non-mouse</code>.</p> <p>The method on <code>simple-pane</code> sets the internal value that is used by <code>scroll-if-not-visible-p</code> on <code>simple-pane</code>.</p>	
See also	<code>simple-pane</code>	

## 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 &amp;optional format =&gt; 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.



Signature	<code>set-clipboard <i>self value</i> &amp;optional <i>string plist</i> =&gt; <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 > Preferences... > Environment > General > 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	<p><code>editor-pane-blink-rate</code></p> <p><code>editor-pane-native-blink-rate</code></p>

## 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	<p><i>prefix</i>      A string or <code>nil</code>.</p> <p><i>suffix</i>      A string or <code>nil</code>.</p> <p><i>child-prefix</i>      A string or <code>nil</code>.</p> <p><i>child-suffix</i>      A string or <code>nil</code>.</p>
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 `:filename-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-editor-parenthesis-colors

*Function*

Summary Sets the colors that are used for parenthesis coloring.

Signature `set-editor-parenthesis-colors colors`

Arguments `colors` A list of colors, `t` or `nil`.

Description The function `set-editor-parenthesis-colors` sets the colors that are used for parenthesis coloring in an `editor-pane` in Lisp mode.

If `colors` is a non-`nil` list, each of its elements must be a valid color specification or a defined color alias. See "The Color System" in the *LispWorks CAPI User Guide* for information about colors.

If it is called when CAPI is running, `set-editor-parenthesis-colors` checks that the colors are valid. If it is called when CAPI is not running, `set-editor-parenthesis-colors` does not check the colors, and a bad color will cause an error later. The colors have an effect only on coloring that happens after the call.

If `colors` is `t` or `nil`, parenthesis coloring is switched on or off, without changing the list of colors.

When parenthesis coloring is off, parentheses are drawn like other characters.

See also `editor-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`.

**Notes** If a hint keyword is repeated in *plist*, the first value is used.

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.

**Compatibility note** The function `set-horizontal-scroll-parameters` supersedes the function `set-scroll-range`, which is deprecated and no longer exported.

The call

```
(set-horizontal-scroll-parameters pane
                                :min-range 0
                                :max-range 42)
```

is equivalent to

```
(set-scroll-range pane 42 nil)
```

Example

See the following files:

```
examples/capi/output-panes/scroll-test.lisp
examples/capi/output-panes/scrolling-without-bar.lisp
```

See also

```
scroll
get-horizontal-scroll-parameters
simple-pane
```

## set-interactive-break-gestures

*Function*

Summary

Sets the break gestures on GTK+ and Motif.

Signature

```
set-interactive-break-gestures gestures => result
```

Arguments

*gestures*            A list of gesture specifiers, or `t`

The function `set-interactive-break-gestures` sets the gestures that can be used to break by typing at an interface.

*gestures* is a list of gesture specifiers. A gesture specifier is an object that `sys:coerce-to-gesture-spec` can recognize.

When an interface is created, the break gestures are set such that typing any one of them when the interface is on top causes an "interface break". This means that, if the interface process is busy, it tries to break it. In a Listener tool, it tries to break the REPL. Otherwise it tries to find a process that appears busy, and breaks that. In the LispWorks IDE, if there is no busy process it raises the Process Browser tool. Otherwise it breaks the current process.

`set-interactive-break-gestures` always returns the list of interactive break gestures.

`gestures` can also be `t`, which means do not change the gestures. This is useful to get the current list.

- Notes
1. `set-interactive-break-gestures` has an effect only on GTK+ and Motif.
  2. `set-interactive-break-gestures` has no effect on interfaces that are already created.
  3. On GTK+ the list can be overridden by the resources file as illustrated in `examples/gtk/gtkrc-break-gestures`

## set-object-automatic-resize

*Function*

Summary	Controls automatic resizing of objects on a pinboard.	
Package	<code>capi</code>	
Signature	<code>set-object-automatic-resize <i>object</i> &amp;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> .

*aspect-ratio* A positive real number, `t` or `nil`.

*aspect-ratio-y-weight*

A real number, default value 0.5.

*pinboard*

A `pinboard-layout`, if supplied.

## Description

The function `set-object-automatic-resize` arranges for *object* to be resized and/or re-positioned automatically when *pinboard* is resized, or removes such a setting.

The value of *aspect-ratio* can be `t`, which means use the current aspect ratio of *object* (that is, its height divided by its width).

*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  

$$(/ (/ \textit{calculated-height} \textit{calculated-width}) \textit{aspect-ratio})$$
3. Compute *correction* as  

$$(\textit{expt} \textit{aspect-ratio-ratio} \textit{aspect-ratio-y-weight})$$
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.

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:



Arguments	<i>pane</i>	A <code>rich-text-pane</code> .																		
	<i>selection</i>	A boolean.																		
	<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-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>If <i>attributes-plist</i> is the symbol <code>:default</code> then the default character format of the pane 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> <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> <p>Additionally these <i>attributes-plist</i> keywords are valid on Microsoft Windows only:</p> <table> <tr> <td><code>:strikeout</code></td> <td>A boolean.</td> </tr> <tr> <td><code>:offset</code></td> <td>An integer specifying the vertical offset of characters from the line (a positive value makes them superscript and a negative value makes them subscript).</td> </tr> <tr> <td><code>:protected</code></td> <td>A boolean.</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>:strikeout</code>	A boolean.	<code>:offset</code>	An integer specifying the vertical offset of characters from the line (a positive value makes them superscript and a negative value makes them subscript).	<code>:protected</code>	A boolean.
<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>:strikeout</code>	A boolean.																			
<code>:offset</code>	An integer specifying the vertical offset of characters from the line (a positive value makes them superscript and a negative value makes them subscript).																			
<code>:protected</code>	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 and select a range.

Set the selection to blue:

```
(capi:set-rich-text-pane-character-format
 rtp
 :attributes-plist '(:color :blue))
```

Make it 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 <i>pane attributes-plist</i></code> <code>=&gt; <i>result</i></code>
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> <pre>:alignment      :left, :right or :center. :start-indent   A number setting the indentation. :offset-indent  A number modifying the indentation.</pre>

- `:offset` A number setting the relative indentation of subsequent lines in a paragraph.
- `:right-indent` A number setting the right margin.
- `:tab-stops` A list of numbers.

Additionally this *attributes-list* keyword is valid on Microsoft Windows, only:

- `: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

*Function*

Summary

Sets the primary selection.

Package

`capi`

Signature

`set-selection` *self value* &optional *string plist* => *result*

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 `:image` whose value should be a `image` 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> is that the pasting application doesn't use the primary selection. For instance, Emacs is configurable by the variable <code>interprogram-paste-function</code>.</p>	
See also	<p><code>selection</code> <code>set-clipboard</code></p>	

## set-printer-metrics

*Function*

Summary	Sets the metrics in the given printer.	
Package	<code>capi</code>	
Signature	<code>set-printer-metrics printer &amp;key left-margin top-margin width height</code>	
Description	<p>The function <code>set-printer-metrics</code> sets the left margin and top margin, and the printable width and printable height, of the given printer. Values outside the bounds of the printer will be corrected.</p>	
Example	To set the margins as large as possible:	

```
(let ((metrics (capi:get-printer-metrics printer)))
  (capi:set-printer-metrics printer
    :left-margin 0
    :top-margin 0
    :width
    (capi:printer-metrics-paper-width metrics)
    :height
    (capi:printer-metrics-paper-height metrics)))
```

Actually this sets the margins to the whole paper size, but the printer driver will move these in to take account of the minimum margins of the device.

See also `get-printer-metrics`  
`set-printer-options`  
`print-dialog`

## set-printer-options

*Function*

Summary      Sets various options in the given printer.

Package      `capi`

Signature     `set-printer-options printer &key output-file first-page last-page orientation copies`

Description   The function `set-printer-options` 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 `print-dialog`.

The *printer* argument should be a printer object returned by `current-printer` or `print-dialog`. This *printer* should then be passed to `with-print-job` to print using the options specified.

The keyword arguments control which options are set. If a keyword is not passed then the option remains unchanged.

Values of *output-file* are:

`nil` Print directly to the device.

`t` Print to a file chosen by the user at printing time.

A pathname Print to the file given by pathname.

Values of *first-page* are:

`:all` Print all pages.

A integer Print from this page to the page given by *last-page*.

Values of *orientation* are:

`:landscape` Print in landscape mode.

`:portrait` Print in portrait mode.

Values of *copies*:

A integer The number of copies to print.

Notes Printer objects cannot be reused after changing their options or metrics. Call `current-printer` after `set-printer-options` to get a new printer object containing the latest settings.

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	<code>set-text-input-pane-selection</code> <i>pane start end</i>	
Arguments	<i>pane</i>	A <code>text-input-pane</code> .
	<i>start, end</i>	Bounding indexes for a subsequence of the text of <i>pane</i> .
Description	The function <code>set-text-input-pane-selection</code> sets the selection in <i>pane</i> to be the text bounded by the indexes <i>start</i> (inclusive) and <i>end</i> (exclusive).	
See also	<code>text-input-pane-selection</code> <code>text-input-pane</code>	

## **set-top-level-interface-geometry**

*Generic 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 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

```
(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))
```

See also

```
top-level-interface-p
top-level-interface-geometry
top-level-interface-display-state
interface
```

## set-vertical-scroll-parameters

*Generic Function*

Summary Allows programmatic control of the parameters of a vertical scroll bar.

Package `capi`

Signature `set-vertical-scroll-parameters` *self* &key *min-range*  
*max-range* *slug-position*  
*slug-size* *page-size* *step-size*

Description The function `set-vertical-scroll-parameters` sets the specified parameters of the vertical 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.

<i>max-range</i>	The maximum data coordinate.
<i>slug-position</i>	The current scroll position.
<i>slug-size</i>	The length of the scroll bar slug.
<i>page-size</i>	The scroll page size.
<i>step-size</i>	The scroll step size.

Compatibility note The function `set-vertical-scroll-parameters` supersedes the function `set-scroll-range`, which is deprecated and no longer exported.

The call

```
(set-vertical-scroll-parameters pane
                               :min-range 0
                               :max-range 42)
```

is equivalent to

```
(set-scroll-range pane nil 42)
```

Example See the following CAPI example files:  
`examples/capi/output-panes/scroll-test.lisp`  
`examples/capi/output-panes/scrolling-without-bar.lisp`

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	<code>shell-pane-command</code>
Description	<p>The class <code>shell-pane</code> creates an editor in which a subprocess runs.</p> <p>User input is interpreted as input to the subprocess. In particular, when the user enters <code>Return</code> in the last line, the line is sent to the subprocess. The output of the subprocess is displayed in the pane.</p> <p>The default value of <code>command</code> is <code>nil</code>, which means that the actual command is determined as follows:</p> <p>On Unix/Linux and Mac OS X, the value of the environment variable <code>ESHELL</code> is used if set, and otherwise the environment variable <code>SHELL</code> is consulted. If that is not set, then <code>/bin/csh</code> (<code>/bin/sh</code> on SVR4 platforms) is run.</p> <p>On Microsoft Windows 98/ME, <code>command</code> is run.</p> <p>On Windows 2000/XP/Vista/Windows 7, <code>cmd</code> is run.</p>
Example	<p>This function emulates user input on <i>pane</i>:</p> <pre>(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)))</pre> <p>This function trampolines to <code>send-keys-to-pane-aux</code> on the right process:</p> <pre>(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))))</pre>

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 <i>pane</i></code>
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 <i>pane</i> =&gt; <i>pane</i></code>
Arguments	<i>pane</i> An instance of <code>simple-pane</code> or a subclass.
Description	The function <code>show-pane</code> restores the pane <i>pane</i> 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	<code>layout</code> <code>row-layout</code> <code>column-layout</code>

**simple-network-pane***Class*

Summary	A graph pane which arranges its nodes in a grid.
Package	<code>capi</code>
Superclasses	<code>graph-pane</code>
Initargs	<code>:x-gap</code> The horizontal node spacing. <code>:y-gap</code> The vertical node spacing.
Description	The class <code>simple-network-pane</code> 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`.

Example 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

<code>:enabled</code>	A boolean controlling whether the pane is enabled.
<code>:background</code>	The background color of the pane.
<code>:foreground</code>	The foreground color of the pane.
<code>:font</code>	The default font for the pane.

<code>:horizontal-scroll</code>	<code>t</code> , <code>:without-bar</code> , or <code>nil</code> . If true the pane can scroll horizontally.
<code>:vertical-scroll</code>	<code>t</code> , <code>:without-bar</code> , or <code>nil</code> . If true the pane can scroll vertically.
<code>:visible-border</code>	A boolean or a keyword controlling whether the pane has a border, for some pane classes.
<code>:internal-border</code>	A non-negative integer, or <code>nil</code> . Controls the width of the internal border.
<code>:cursor</code>	A keyword naming a built-in cursor, or a cursor object, or <code>nil</code> .
<code>:pane-menu</code>	Specifies a menu to be raised by the <code>:post-menu</code> gesture.
<code>:drop-callback</code>	Specifies a drop callback for <code>output-pane</code> or <code>interface</code> (and on Cocoa, <code>list-panel</code> or <code>tree-view</code> ).
<code>:drag-callback</code>	Specifies a drag callback for <code>list-panel</code> or <code>tree-view</code> .
<code>:scroll-if-not-visible-p</code>	Defines whether, when the focus is given to the pane and the pane is not fully visible, the pane's parent is automatically scrolled to show it.
<code>:toolbar-title</code>	A string.

Accessors	<pre> simple-pane-enabled simple-pane-background simple-pane-foreground simple-pane-font simple-pane-cursor simple-pane-scroll-callback simple-pane-drop-callback simple-pane-drag-callback </pre>								
Readers	<pre> simple-pane-horizontal-scroll simple-pane-vertical-scroll simple-pane-visible-border </pre>								
Description	<p><i>enabled</i> determines whether the pane is enabled. The default value is <code>t</code>. Note that changing the enabled state of a visible pane changes its appearance.</p> <p><i>background</i> and <i>foreground</i> are colors specified using the Graphics Ports color system. Additionally on Cocoa, the special value <code>:transparent</code> is supported, which makes the pane's background match that of its parent.</p> <p><i>font</i> should be <code>font</code>, a font description, or <code>nil</code>.</p> <p>The value for <i>visible-border</i> can be any of the following, with the stated meanings where applicable:</p> <table> <tr> <td><code>nil</code></td> <td>Has no border.</td> </tr> <tr> <td><code>t</code></td> <td>Has a border.</td> </tr> <tr> <td><code>:default</code></td> <td>Use the default for the window type.</td> </tr> <tr> <td><code>:outline</code></td> <td>Add an outline border.</td> </tr> </table> <p>There are various platform/pane class combinations which do not respond to all values of <i>visible-border</i>. For instance, on Windows XP with the default theme, <code>text-input-choice</code> and <code>option-pane</code> always have a visible border regardless of the value of <i>visible-border</i>, while other classes including <code>display-pane</code>, <code>text-input-pane</code>, <code>list-panel</code>, <code>editor-pane</code> and <code>graph-pane</code> have three distinct border styles, with <i>visible-border</i> <code>:default</code> meaning the same as <i>visible-border</i> <code>t</code>.</p>	<code>nil</code>	Has no border.	<code>t</code>	Has a border.	<code>:default</code>	Use the default for the window type.	<code>:outline</code>	Add an outline border.
<code>nil</code>	Has no border.								
<code>t</code>	Has a border.								
<code>:default</code>	Use the default for the window type.								
<code>:outline</code>	Add an outline border.								

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
<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

Table 1.2

<i>cursor</i>	Cocoa	Windows	Motif
<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. On Cocoa, `list-panel` and `tree-view` also support *drop-callback*. 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 which is an `output-pane` or `interface` (not `tree-view` or `list-panel`). 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 while the user is dragging 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. For `output-pane`, it can use `drop-object-pane-x` and `drop-object-pane-y` to query the mouse position relative to the pane. For `list-panel` and `tree-view`, it can use `drop-object-collection-index` or `drop-object-collection-item` to query where the user is attempting to drop the object and can call their `setf` functions to adjust this position. 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. For `output-pane` and `interface`, 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 the `:drag` stage, 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. It should also update the pane to incorporate the object in whatever way the application requires.

*drag-callback* can be specified for a pane that is an instance of `list-panel` or `tree-view`. The default value of *drag-callback* is `nil`, which means that there is no support for dragging from the pane. Otherwise, it should be a function designator with this signature:

```
drag-callback pane info => plist
```

When the user drags items in the pane, the CAPI calls the *drag-callback*. *pane* is the pane itself and *info* is a list of item indices that are being dragged (compare with `choice-selection`). The *drag-callback* should return the plist *plist* whose keys are the data formats to be dragged, with a value associated with each format. Formats are arbitrary keywords that must be interpreted by the pane where you intend to drop the values (see the *drop-callback*). The `:string` format is understood by some other panes that expect text.

*scroll-if-not-visible-p* controls scrolling behavior of the parent when the pane is given the input focus. *scroll-if-not-visible-p* can be `t`, `nil`, or `:non-mouse`. See `scroll-if-not-visible-p` for details. When this initarg is supplied, the generic function (`setf scroll-if-not-visible-p`) is called with it.

If the pane is used in the *toolbar-items* list of an *interface*, then *toolbar-title* should be a short string that will be shown near to the pane if required for the toolbar.

## Notes

1. *foreground* is ignored for buttons on Windows and Cocoa.
2. 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`.

```

Example  (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)

```

For an example illustrating the use of *drag-callback*, see `examples/capi/choices/drag-and-drop.lisp`

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	<code>simple-pane-visible-size <i>pane</i> =&gt; <i>width</i>, <i>height</i></code>	
Arguments	<code><i>pane</i></code>	A simple pane.
Values	<code><i>width</i></code>	The width of the visible part of <i>pane</i> , or <code>nil</code> .
	<code><i>height</i></code>	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	<code>simple-pane-visible-height</code> <code>simple-pane-visible-width</code> <code>with-geometry</code>	

## simple-pane-visible-width

## Generic Function

Summary	Gets the visible width of a pane.	
Package	<code>capi</code>	
Signature	<code>simple-pane-visible-width <i>pane</i> =&gt; <i>result</i></code>	
Arguments	<code><i>pane</i></code>	A simple pane.
Values	<code><i>result</i></code>	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>	

See the *LispWorks CAPI User Guide* for a description of the visible size of a pane.

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/Setup.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** `capi`

**Superclasses** `range-pane`  
`titled-object`  
`simple-pane`

**Initargs** `:show-value-p` A generalized boolean.  
`:start-point`

A keyword.

**Readers** `slider-show-value-p`  
`slider-start-point`

**Description** The `slider` class allows the user to enter a number by moving a marker on a sliding scale to the desired value. *show-value-p* determines whether the slider displays the current value. The default value is `t`.

**Note:** *show-value-p* is ignored on Microsoft Windows and Cocoa.

*start-point* specifies which end of the slider is the start point in the range. The values allowed depend on the *orientation* of the slider. For horizontal sliders, *start-point* can take these values:

`:left`            The start point is on the left.  
`:right`            The start point is on the right.  
`:default`          The start point is at the default side (the left).

For vertical sliders, *start-point* can take these values:

`:top`              The start point is at the top.  
`:bottom`            The start point is at the bottom.  
`:default`          The start point is at the default position,  
which is the top on Microsoft Windows and  
Motif, and the bottom on Cocoa.

## 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 items =&gt; 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 new-sort-type &amp;key 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	<p>The generic function <code>sorted-object-sort-by</code> sets the sort type of <i>pane</i> to <i>new-sort-type</i>.</p> <p><i>new-sort-type</i> must match the type of one of the sort descriptions of <i>pane</i>.</p> <p>If <i>allow-reverse</i> is non-nil 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>.</p> <p>If <i>pane</i> is a <code>list-panel</code>, then <code>sorted-object-sort-by</code> also calls <code>sort-object-items-by</code> to sort the items with the new sort type. For your own subclasses of <code>sorted-object</code> which are not subclasses of <code>list-panel</code>, if you need this behavior define an <code>:after</code> method that calls <code>sort-object-items-by</code>. You can also define <code>:after</code> methods on subclasses of <code>list-panel</code> to perform other tasks each time the items are sorted.</p>
See also	<p><code>sort-object-items-by</code>  <code>sorted-object</code></p>

## start-gc-monitor

*Function*

Summary	Starts a Lisp Monitor window.	
Package	<code>capi</code>	
Signature	<code>start-gc-monitor screen =&gt; result</code>	
Arguments	<i>screen</i>	A screen.
Values	<i>result</i>	A boolean.
Description	<p>The function <code>start-gc-monitor</code> starts a Lisp Monitor window (otherwise known as the GC or Garbage Collector monitor) on the screen <i>screen</i>.</p>	

*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. There is no Lisp Monitor window on other platforms.

On Motif, `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`

## static-layout

*Class*

**Summary** A layout that allows its children to be positioned anywhere within itself.

**Package** `capi`

**Superclasses** `layout`

**Subclasses** `pinboard-layout`

**Initargs** `:fit-size-to-children`

A generalized boolean.

**Description** The class `static-layout` is a layout that allows its children to be positioned anywhere within itself.

When a `static-layout` lays out its children, it positions them at the *x* and *y* specified as hints (using `:x` and `:y`), and sizes them to their minimum size (which can be specified using `:visible-min-width` and `:visible-max-width`).

If `fit-size-to-children` is true, the `static-layout` is made sufficiently large to accommodate all of its children, and grows if necessary when a child is added. This is the default behavior. Otherwise the static 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 `simple-layout` which surrounds a single child, and adopts the size constraints of that child.

**Example** Here is an example of a static 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)
```

See also `pinboard-layout`

## stop-gc-monitor

*Function*

**Summary** Stop a Lisp Monitor.

**Package** `capi`

**Signature** `stop-gc-monitor screen => result`

**Arguments** `screen` A screen.

**Values** `result` A boolean.

**Description** The function `stop-gc-monitor` stops the Lisp Monitor window on the screen `screen`.

*result* is `t` if it stopped a Lisp monitor, and `nil` if there was no Lisp monitor running on *screen*.

Note that this works only on Motif. The Lisp monitor can be restarted with `start-gc-monitor`.

See also `start-gc-monitor`

## **stop-sound** *Function*

Summary Stops a sound from playing.

Signature `stop-sound sound`

Arguments *sound* A sound object returned by `load-sound`.

Description The function `stop-sound` stops the sound *sound* from playing.

See also `play-sound`

## **switchable-layout** *Class*

Summary A subclass of `simple-layout` 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 `capi`

Superclasses `simple-layout`

Initargs `:visible-child`

The currently visible pane from the children.

`:combine-child-constraints`

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-nil. 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-pane1` 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`.

### Example

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 <i>tab-layout</i> , 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</code> <i>tab-layout</i> => <i>result</i>	
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.
Notes	The <code>text-input-pane</code> initarg value <i>enabled</i> <code>:read-only</code> is not supported for <code>text-input-choice</code> on Microsoft Windows.
Examples	See <code>examples/capi/elements/text-input-choice.lisp</code> .
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	<code>:text</code> The text in the pane.  <code>:caret-position</code> The position of the caret in the text (from 0).  <code>:max-characters</code> The maximum number of characters allowed.  <code>:enabled</code> Controls the enabled state of the pane.  <code>:completion-function</code> A function called to complete the text.  <code>:in-place-completion-function</code> A function designator.  <code>:file-completion</code> t, nil or a pathname designator.  <code>:in-place-filter</code> A boolean.  <code>:directories-only</code> A boolean.  <code>:ignore-file-suffices</code> A list of strings or the keyword <code>:default</code> .  <code>:callback-type</code> The type of arguments to <i>callback</i> .  <code>:callback</code> A function usually called when the user presses Return.  <code>:change-callback-type</code> The type of arguments to <i>callback</i> .

<code>:change-callback</code>	A function called when a change is made.
<code>:confirm-change-function</code>	A function called to validate a change. Note: Implemented for Motif only, not Microsoft Windows or Mac OS X.
<code>:navigation-callback</code>	A function called when certain keyboard gestures occur in the pane.
<code>:editing-callback</code>	A function called when editing starts or stops.
<code>:gesture-callbacks</code>	A list of pairs ( <i>gesture</i> . <i>callback</i> ).
<code>:complete-do-action</code>	A boolean.
<code>:text-change-callback</code>	A function designator.
<code>:buttons</code>	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`.

**Note:** `enabled :read-only` is not supported for the subclass `text-input-choice` on Microsoft Windows.

A `completion-function` can be specified which will get called when the completion gesture is made by the user (by pressing the `tab` key) or when `text-input-pane-complete-text` is called. The function should have signature:

```
completion-function pane string => completions, start, end
```

where `pane` is the `text-input-pane` itself and `string` is the string to complete. When completion is invoked `completion-function` is called with `pane` and a string containing the text of pane to the left of the cursor.

The `completion-function` is called with the pane and the text to complete and should return either `nil`, the completed text as a string or a list `completions` 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. In addition, the `completion-function` can return two more values, `start` and `end`, which specify a range in the text that is to be replaced if the completion is successful.

*in-place-completion-function* tells the pane to do in-place completion and specifies the function to use. The function should have signature:

```
in-place-completion-function pane string => completions,  
start, end
```

where *pane* is the `text-input-pane` itself and *string* is the string to complete. When in-place completion is invoked *in-place-completion-function* is called with *pane* and a string containing the text of *pane* to the left of the cursor.

*completions* needs to be a list of strings that are possible completions, a single string that is a unique completion, or the symbol `:destroy`. `:destroy` means that the in-place completion needs to stop and close the in-place window. In addition, the completion function can return two more values, *start* and *end*, which specify a range in the text that is to be replaced if the completion is successful. The function is called repeatedly whenever there is a change to the text that should be completed.

**Note:** If *in-place-completion-function* needs some dynamic information, it can put it in a property of the pane (using `capl-object-property`).

**Note:** The `initarg :file-completion` overrides `:in-place-completion-function`.

**Note:** The in-place completion mechanism uses *gesture-calls* to implement the functionality.

**Note:** `:in-place-filter` can be used to specify that the in-place window can have a filter.

See "In-place completion" in the *LispWorks CAPI User Guide* for the user interaction.

*file-completion*, if non-nil, tells the pane to do file completion using an in-place window. See "In-place completion" in the *LispWorks CAPI User Guide* for the interaction.

If *file-completion* is a pathname designator, its location is used as the root path for the completion.

**Note:** `:file-completion` overrides `:in-place-completion-function`.

**Note:** The behavior of in-place completion is somewhat different from other completion.

**Note:** The initargs `:directories-only` and `:ignore-file-suffices` can be used to change the behavior of the completion.

The default value of *file-completion* and *in-place-completion-function* is `nil`.

*in-place-filter* takes effect only when either *in-place-completion-function* or *file-completion* is non-`nil`. If *in-place-filter* is `t` then the in-place window can have a filter. Note that the filter needs to be requested by a user gesture. `Control+Return` is the default in-place filter gesture. The default value of *in-place-filter* is `t`.

*directories-only* takes effect only if *file-completion* is used. If *directories-only* is `t` then in-place completion shows only directories. The default value of *directories-only* is `nil`.

*ignore-file-suffices* takes effect only if *file-completion* is used. It tells in-place completion to ignore files whose file namestring (the result of `cl:file-namestring`) ends with any of the strings in the list *ignore-file-suffices*. If *ignore-file-suffices* is `:default`, then completion uses the default value, which is the value of `editor:*ignorable-file-suffices*` (see `config/a-dot-lispworks.lisp`).

*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.

*gesture-callbacks* provides callbacks to perform for specific keyboard gestures. Each *gesture* must be an object that `sys:coerce-to-gesture-spec` can coerce to a `gesture-spec`. Each *callback* can be a callable (symbol or function) which takes one argument, the pane. Alternatively each *callback* can be a list of the form (*function arguments*). Note that in this case, the pane itself is not automatically passed to the *function* amongst *arguments*.

When the user enters a gesture that matches *gesture* in any pair amongst *gesture-callbacks*, the *callback* is executed and the gesture is not processed any more.

**Note:** The interaction of in-place completion is implemented using *gesture-callbacks*. Gestures which you define explicitly by *gesture-callbacks* override the gestures which are defined implicitly by the in-place completion mechanism.

**Note:** For gestures that change the text, *text-change-callback* is probably better than *gesture-callbacks*.

When *complete-do-action* is non-nil, completion of the text in the pane automatically invokes *callback* (if *callback* is non-nil). The default value of *complete-do-action* is `nil`.

*text-change-callback* is a change callback (see *change-callback*) that is called only when the text in the pane changes. In contrast, *change-callback* is also called when the caret moves. If both *text-change-callback* and *change-callback* are supplied, only *text-change-callback* is invoked.

*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 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 true, 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.

<code>:orientation</code>	The value is either <code>:horizontal</code> or <code>:vertical</code> . <i>orientation</i> controls the orientation of the toolbar. This is useful for <code>multi-line-text-input-pane</code> . The default value is <code>:horizontal</code> .
<code>:adjust</code>	The value is <code>:top</code> , <code>:center</code> , <code>:centre</code> or <code>:bottom</code> . <i>adjust</i> controls how the buttons are adjusted vertically relative to the text input pane. This is useful for <code>multi-line-text-input-pane</code> . The default value is <code>:center</code> .
<code>:position</code>	The value is <code>:top</code> , <code>:bottom</code> , <code>:left</code> or <code>:right</code> . <i>position</i> determines whether the buttons appear above, below, left or right of the text input pane. If <code>:position</code> 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:

<code>:enabled</code>	A value that controls whether the button is enabled. (See the reader <code>text-input-pane-buttons-enabled</code> ).
<code>:image</code>	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 <code>register-image-translation</code> , an <code>image</code> object as returned by <code>load-image</code> or an <code>external-image</code> . The default image is one of the symbols <code>ok-but-</code>

ton, cancel-button or complete-button, which are pre-registered 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.

Example

```
(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*))))
```

This example illustrates the use of *gesture-callbacks*. `ctrl+e` moves the cursor to the end of the input, `ctrl+a` moves it to the start, and `ctrl+6` does something else:

```

(capi:contain
 (make-instance
  'capi:text-input-pane
  :gesture-callbacks
  (list
   (cons
    #\Ctrl-\e
    #'(lambda (tip)
         (setf (capi:text-input-pane-caret-position tip)
               (length (capi:text-input-pane-text
                        tip))))))
   (cons
    #\Ctrl-\a
    #'(lambda (tip)
         (setf (capi:text-input-pane-caret-position tip)
               0)))
   (cons
    #\Ctrl-6 'do-something-else))))

```

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 <i>completion-function</i> in a <code>text-input-pane</code> .	
Package	<code>capi</code>	
Signature	<code>text-input-pane-complete-text</code> <i>pane</i> => <i>result</i>	
Arguments	<i>pane</i>	A <code>text-input-pane</code> .

Values	<i>result</i>	A string, or <code>nil</code> .
Description	<p>The function <code>text-input-pane-complete-text</code> calls the <i>completion-function</i> of <i>pane</i> with the current <i>text</i>. If this call is successful, then the <i>text</i> of <i>pane</i> is set to the result, and <code>text-input-pane-complete-text</code> returns this result. Otherwise, <i>result</i> is <code>nil</code>.</p> <p>Note: the <i>completion-function</i> may return a list of completion candidates, in which case <code>text-input-pane-complete-text</code> prompts the user to select one of the candidates.</p>	
See also	<code>text-input-pane</code>	

## 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	<p>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.</p>	
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 <code>text-input-pane-cut</code> performs the clipboard cut 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-copy</code> <code>text-input-pane-delete</code> <code>text-input-pane-paste</code>

**text-input-pane-delete***Function*

Summary	Deletes the selected text in a <code>text-input-pane</code> .
Package	<code>capi</code>
Signature	<code>text-input-pane-delete</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-delete</code> deletes the selected text in <i>text-input-pane</i> . 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-in-place-complete**

*Function*

Summary      Raises the non-focus completion window.

Signature      `text-input-pane-in-place-complete` *text-input-pane*

Arguments      *text-input-pane*    A `text-input-pane`

Description      The function `text-input-pane-in-place-complete` raises the non-focus completion window.

The pane *text-input-pane* must have been made with either *in-place-completion-function* or *file-completion*. See the description of this functionality in `text-input-pane`.

See also      `text-input-pane`

## **text-input-pane-paste**

*Function*

Summary      Pastes the clipboard text into a `text-input-pane`.

Package      `capi`

Signature      `text-input-pane-paste` *text-input-pane*

Arguments      *text-input-pane*    An instance of `text-input-pane` or a subclass.

Description      The function `text-input-pane-paste` performs the clipboard paste operation on *text-input-pane*, replacing any selected text.

See also            `clipboard`  
                       `text-input-pane`  
                       `text-input-pane-selection`  
                       `text-input-pane-cut`  
                       `text-input-pane-copy`  
                       `text-input-pane-delete`

## text-input-pane-selected-text

*Function*

Summary            Returns the selected text in a `text-input-pane`.

Package            `capi`

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

Arguments          *text-input-pane*    An instance of `text-input-pane` or a subclass.

Values              *result*              A string or `nil`.

Description        The function `text-input-pane-selected-text` returns the selected text in *text-input-pane*, or `nil` 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	<code>capi</code>
Signature	<code>text-input-pane-selection pane =&gt; start, end</code>
Arguments	<i>pane</i> A <code>text-input-pane</code> .
Values	<i>start, end</i> Non-negative integers.
Description	<p>The function <code>text-input-pane-selection</code> returns as multiple values the bounding indexes of the selection in <i>pane</i>. That is, <i>start</i> is the inclusive index of the first selected character, and <i>end</i> is one greater than the index of the last selected character.</p> <p>If there is no selection, then both <i>start</i> and <i>end</i> are the caret position in <i>pane</i>.</p>
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-p</code>

## **text-input-pane-selection-p**

*Function*

Summary	Returns true if there is selected text in a <code>text-input-pane</code> .
Package	<code>capi</code>
Signature	<code>text-input-pane-selection-p pane =&gt; selectionp</code>
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 `set-text-input-pane-selection`  
`text-input-pane`  
`text-input-pane-selected-text`  
`text-input-pane-selection`

## text-input-range

*Class*

**Summary**      The class `text-input-range` 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**      `cap`

**Superclasses**    `titled-object`  
`simple-pane`

**Initargs**

- `:start`      An integer specifying the lowest possible value in the range.
- `:end`        An integer specifying the highest possible value in the range.
- `:wraps-p`     A generalized boolean.
- `:value`      An integer specifying the current value in the pane.
- `:callback`    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)`.

Example

```
(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     capi

Superclasses

- `titled-object`
- `simple-pane`

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>
Example	<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	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.  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> .  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> .
Example	<pre>(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)))))</pre>

**titled-object***Class*

Summary	The class <code>titled-object</code> 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	<code>capi</code>	
Subclasses	<pre>interface layout title-pane display-pane text-input-pane toolbar button-panel list-panel option-pane progress-bar output-pane slider</pre>	
Initargs	<code>:title</code>	A title string for the pane (or <code>nil</code> ).
	<code>:title-args</code>	Initargs to the title <code>make-instance</code> .
	<code>:title-font</code>	The font used for the title.
	<code>:title-position</code>	The position of the title.
	<code>:title-adjust</code>	How to adjust the title relative to the pane.
	<code>:title-gap</code>	The gap between the title and the pane.
	<code>:message</code>	A message string for the pane (or <code>nil</code> ).
	<code>:mnemonic-title</code>	A string specifying the title and a mnemonic. Applies only to the subclasses specified below.
	<code>:message-gap</code>	The gap between the message and the pane.

Accessors	<pre>titled-object-title titled-object-title-font titled-object-message titled-object-message-font</pre>										
Description	<p>The titled pane makes its title decoration from a <code>title-pane</code> and the message decoration from a <code>message-pane</code>.</p> <p>The <i>text</i> of the <code>title-pane</code> is passed via the <code>titled-object</code> initarg <i>title</i> and the <i>text</i> of the <code>message-pane</code> is passed via the <code>titled-object</code> initarg <i>message</i>.</p> <p>The initargs and font for the <code>title-pane</code> are passed via the <code>titled-object</code> initargs <i>title-args</i> and <i>title-font</i> respectively.</p> <p><i>title-gap</i> specifies the size in pixels of the gap between the title and the pane. The default value of <i>title-gap</i> is 3.</p> <p>For subclasses other than <code>interface</code>, the font used for the <i>message</i> can be found by <code>titled-object-message-font</code> and set by <code>(setf titled-object-message-font)</code>.</p> <p><i>message-gap</i> specifies the size in pixels of the gap between the message and the pane. The default value of <i>message-gap</i> is 3.</p> <p>The message is always placed below the pane, but the title's position can be adjusted by specifying <i>title-position</i> which can be any of the following.</p> <table> <tr> <td><code>:left</code></td> <td>Place the title to the left of the pane.</td> </tr> <tr> <td><code>:right</code></td> <td>Place the title to the right of the pane.</td> </tr> <tr> <td><code>:top</code></td> <td>Place the title above the pane.</td> </tr> <tr> <td><code>:bottom</code></td> <td>Place the title below the pane.</td> </tr> <tr> <td><code>:frame</code></td> <td>Place the title in a frame (like a groupbox) around the pane.</td> </tr> </table> <p>The <i>title-adjust</i> slot is used to adjust the title so that it is left justified, right justified or centered. The value of <i>title-adjust</i> can be any of the values accepted by the function <code>pane-adjusted-offset</code>, which are <code>:left</code>, <code>:right</code>, <code>:top</code>, <code>:bottom</code>, <code>:center</code> and <code>:centre</code>.</p>	<code>:left</code>	Place the title to the left of the pane.	<code>:right</code>	Place the title to the right of the pane.	<code>:top</code>	Place the title above the pane.	<code>:bottom</code>	Place the title below the pane.	<code>:frame</code>	Place the title in a frame (like a groupbox) around the pane.
<code>:left</code>	Place the title to the left of the pane.										
<code>:right</code>	Place the title to the right of the pane.										
<code>:top</code>	Place the title above the pane.										
<code>:bottom</code>	Place the title below the pane.										
<code>:frame</code>	Place the title in a frame (like a groupbox) around the pane.										

**Note:** *title-adjust* cannot handle both *x* and *y*. It is designed for cases like this:

```
(capi:contain
  (make-instance 'capi:list-panel
    :items '(1 2 3 4 5)
    :title "Temp"
    :title-position :left
    :title-adjust :center
    :title-args
    '(:visible-min-width (:character 12))))
```

*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`.

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.

Example

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)))

```

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` initarg *title-position*. The values `:top`, `:bottom`, `:left` and `:right` are allowed.

### Example

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`.

Notes `text-input-pane`, `option-pane`, and `text-input-choice` and so on cannot contain titles when embedded in a `toolbar`.

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`.

In addition, an interface can have its own toolbar buttons, specified by its *toolbar-items*. There is no toolbar object in that situation.

*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:

<code>:button</code>	There is a separate smaller button for the dropdown menu next to the main button.
<code>:only</code>	There is no main button, only the smaller button for the dropdown.
<code>:delayed</code>	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 <i>callback</i> is called.

**Note:** *dropdown-menu-kind* is not supported for toolbar buttons in the `interface toolbar-items` list.

*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`.

**Note:** *popup-interface* is not supported for toolbar buttons in the `interface toolbar-items` list.

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 "accelerator-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`.

## Example

See `examples/capi/elements/toolbar.lisp`.

## 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

*Generic 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"> <li><code>:normal</code>      The window is visible and has its normal size.</li> <li><code>:maximized</code>    The window is visible and has been maximized.</li> <li><code>:iconic</code>        The window is visible as an icon.</li> <li><code>:hidden</code>        The window is not visible.</li> </ul>

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

*Generic Function*

Summary	Returns the geometry of the top level interface.
Package	<code>capi</code>
Signature	<code>top-level-interface-geometry <i>interface</i></code>
Description	The generic function <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)))</pre>

```
(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))

;; 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 <i>interface</i> =&gt; <i>key</i>, <i>product-name</i></code>	
Arguments	<code>interface</code>	A top level interface.
Values	<code>key</code>	A symbol.
	<code>product-name</code>	A symbol, a string or a list of strings.

## Description

The generic function `top-level-interface-geometry-key` returns as multiple values a key and a product name, which determine where the geometry of *interface* is saved. The saved geometry is used when displaying a future instance.

The supplied method on `interface` returns the class name of *interface* as the *key*, and `nil` as the *product-name*. 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*

Summary      The predicate for top level interfaces.

Package      `capi`

Signature     `top-level-interface-p pane`

Description   The generic function `top-level-interface-p` returns true if *pane* is a top level interface.

See also      `top-level-interface`  
`top-level-interface-geometry`  
`top-level-interface-display-state`  
`interface`  
`element`

## top-level-interface-save-geometry-p

*Generic Function*

Package      `capi`

Signature     `top-level-interface-save-geometry-p interface => result`

Description   The generic function `top-level-interface-save-geometry-p` returns true if the geometry of *interface* should be saved for use by a future instance.

The default method (on `interface`) returns `nil`.

See also      `top-level-interface-geometry-key`

## tracking-pinboard-layout

*Class*

Summary      A pinboard with automatic highlighting.

Package      `capi`

Superclasses `pinboard-layout`

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`, or `t`.

An integer specifies the default initial status, and `t` means the same as 2 (that is, by default the checkboxes are checked initially).

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 descendants 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. Use `extended-selection-tree-view` if you need other selection interaction styles.

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 descendants 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 descendants have the same status. If an item is grey-checked, then its descendants have various statuses. When the status of an item changes, all the descendants of that item change 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`.

## Notes

1. 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.
2. Although `tree-view` is a subclass of `collection`, it does its own items handling and you must not access its *items* and related slots directly. In particular for `tree-view` do not pass `:items`, `:items-count-function`, `:items-get-function` OR `:items-map-function`, and do not use the corresponding accessors.

## See also

```
choice  
extended-selection-tree-view  
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</code> <i>tree-view item</i>	
Arguments	<i>tree-view</i>	A tree view.
	<i>item</i>	A displayed item of <i>tree-view</i> .
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 <i>item</i> must be an item that is displayed in <i>tree-view</i> .	
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</code> <i>tree-view item</i>	
Signature	<code>(setf tree-view-expanded-p)</code> <i>on tree-view item</i>	
Arguments	<i>tree-view</i>	A <code>tree-view</code> .

*item*            An item.  
*on*                A boolean.

Description      The generic function `tree-view-expanded-p` is the predicate for whether *item* is expanded in *tree-view*. If *item* is not in *tree-view*, the function returns `nil`.

`(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        *tree-view*        A tree view.  
*item*                An item.  
*status*             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 `capi`

Signature `tree-view-item-children-checkbox-status` *tree-view item*  
`=>` *result*

Arguments *tree-view* A `tree-view`.  
*item* An item.

Values *result* A list of conses (*child . status*) where each *child* is a child of *item* and *status* is *child's* checkbox status.

Description The function `tree-view-item-children-checkbox-status` returns *item's* children together with their checkbox statuses. Note that, if *item* has not been expanded in *tree-view*, then it has no children and *result* will be `nil`.

See also `tree-view`  
`tree-view-item-checkbox-status`

## `tree-view-update-an-item` *Generic Function*

Summary Updates an item in a `tree-view`.

Package `capi`

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	<code>tree-view</code> <code>tree-view-update-item</code>

## tree-view-update-item

## Generic Function

Summary	Updates an item in a <code>tree-view</code> .	
Package	<code>capi</code>	
Signature	<code>tree-view-update-item</code> <i>tree-view item in-parent</i>	
Arguments	<i>tree-view</i>	A <code>tree-view</code> .
	<i>item</i>	An item.
	<i>in-parent</i>	A boolean.
Description	The generic function <code>tree-view-update-item</code> updates the item <i>item</i> in <i>tree-view</i> . This includes recomputing the text, images and children of <i>item</i> . This is useful when the data in <i>tree-view</i> changes, but the entire tree does not need recomputing.  When <i>in-parent</i> is non-nil, <code>tree-view-update-item</code> updates the children of the parent of <i>item</i> . This is useful when <i>item</i> is actually removed from <i>tree-view</i> , causing the children of its parent to be re-positioned.	
See also	<code>tree-view</code>	

**undefine-menu***Macro*

Package	<code>capi</code>
Signature	<code>undefine-menu <i>function-name</i> &amp;rest <i>args</i></code>
Description	This function undefines a menu created with <code>define-menu</code> .
See also	<code>define-menu</code> <code>menu</code>

**unhighlight-pinboard-object***Generic Function*

Summary	Removes the highlighting from a <code>pinboard-object</code> .
Package	<code>capi</code>
Signature	<code>unhighlight-pinboard-object <i>pinboard object</i> &amp;key <i>redisplay</i></code>
Description	This removes the highlighting from a pinboard object if necessary, and then if <i>redisplay</i> is non-nil it redisplay it. The default value of <i>redisplay</i> is <code>t</code> .  To highlight a pinboard object use <code>highlight-pinboard-object</code> .
See also	<code>highlight-pinboard-object</code> <code>pinboard-object</code>

**uninstall-postscript-printer***Function*

Summary	Uninstalls a Postscript printer definition.
Package	<code>capi</code>

Signature	<code>uninstall-postscript-printer</code> <i>name</i> &key <i>if-does-not-exist</i> <i>deletep</i>
Arguments	<p><i>name</i>           A string.</p> <p><i>if-does-not-exist</i>   One of <code>nil</code> or <code>:error</code>.</p> <p><i>deletep</i>           A boolean.</p>
Description	<p>Uninstalls a PostScript printer definition for the given device <i>name</i>.</p> <p>This applies only on GTK+ and Motif.</p> <p><i>if-does-not-exist</i> controls what happens if the named printer does not exist. The default value is <code>:error</code>.</p> <p><i>deletep</i>, 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 <i>deletep</i> is <code>nil</code>.</p>
See also	<code>install-postscript-printer</code>

## unmap-typeout

*Function*

Package	<code>capi</code>
Signature	<code>unmap-typeout</code> <i>collector-pane</i>
Description	This switches the <i>collector-pane</i> out from its switchable layout, and brings back the pane that was there before <code>map-typeout</code> was called.
See also	<p><code>map-typeout</code></p> <p><code>with-random-typeout</code></p> <p><code>collector-pane</code></p>

**update-all-interface-titles***Function*

Summary	Updates interface window titles.
Package	<code>capi</code>
Signature	<code>update-all-interface-titles</code>
Description	<p>The function <code>update-all-interface-titles</code> can be used to update all the <code>interface</code> window titles when needed.</p> <p>This is useful when <code>interface-extend-title</code> may return a new, different, value.</p> <p><code>update-all-interface-titles</code> calls <code>update-screen-interface-titles</code> on all the screens.</p>
See also	<p><code>interface-extend-title</code></p> <p><code>update-screen-interface-titles</code></p>

**update-interface-title***Generic Function*

Summary	Updates the title of an interface window.
Package	<code>capi</code>
Signature	<code>update-interface-title</code> <i>interface</i>
Arguments	<i>interface</i> A CAPI interface.
Description	<p>The generic function <code>update-interface-title</code> updates the title of interface <i>interface</i>. This is useful when <code>interface-extend-title</code> may return a new, different, value.</p> <p>You can specialize <code>update-interface-title</code> if needed.</p> <p>To update all the interface titles, use <code>update-all-interface-titles</code> OR <code>update-screen-interface-titles</code>.</p>

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** `capi`

**Description** Each function in the list `*update-screen-interfaces-hooks*` is called when an interface *interface* is created or destroyed.

Each function takes two arguments: the screen and *interface*. You should not remove system functions from this variable so take care if setting its value. Only add or delete your own functions.

**update-toolbar**

*Function*

**Summary** Updates a toolbar object.

**Package** `capi`

**Signature** `update-toolbar self`

**Description** The `update-toolbar` function updates the toolbar object *self*. It computes the enabled function of *self* and the enabled functions of any toolbar components or toolbar buttons contained in it. Each toolbar object is enabled if the enabled function returns `t`, and is disabled if it returns `nil`.

See also `toolbar`  
`toolbar-button`  
`toolbar-component`

## **with-atomic-redisplay**

*Macro*

**Summary**      The `with-atomic-redisplay` macro delays the updating of specified panes until all state changes have been performed.

**Package**        `capi`

**Signature**     `with-atomic-redisplay (&rest panes) &body body`

**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 `with-atomic-redisplay` defers all visible changes to the state of each pane in *panes* until the end of the scope of the macro.

**See also**        `display`  
`simple-pane`

## **with-busy-interface**

*Macro*

**Summary**        Displays an alternate cursor during the execution of some code, on platforms other than Cocoa.

**Package**        `capi`

**Signature**     `with-busy-interface (pane &key cursor delay) &body body`

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> specifies a time in seconds 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> <p><code>with-busy-interface</code> must be called in the process of the interface containing <i>pane</i>.</p> <p><code>with-busy-interface</code> has no effect on Cocoa.</p>
See also	<code>simple-pane</code>

## with-dialog-results

*Macro*

Summary	Displays a dialog and executes a body when the dialog is dismissed.	
Package	<code>capi</code>	
Signature	<code>with-dialog-results (&amp;rest <i>results</i>) <i>dialog-form</i> &amp;body <i>body</i> =&gt; :continuation, nil</code>	
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, GTK+ 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.

### Examples

On Microsoft Windows, GTK+ 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)))
```

Here is an example with skeleton code for using `with-dialog-results`. Note that the dialog function (`choose-file` below) that is called by `with-dialog-results` must take a *continuation* keyword argument and pass it to a CAPI prompting function. Also note that the call to the CAPI prompting function must be the last form in the dialog function. Forms after the CAPI prompting function will be executed at an indeterminate time, and their values will not be used in the body of `with-dialog-results`.

```

(defun choose-file (&key continuation)
  (print 'in-choose-file)
  (capi:prompt-for-file "Choose File"
                       :pathname "~/Desktop/"
                       :continuation continuation))

(defun open-file (rep)
  (format t "~%Opening ~a~%" rep))

(defun my-callback ()
  (print 'doing-something-before)
  (capi:with-dialog-results (res ok-p)
    (choose-file)
    (print 'after-choose-file)
    (if ok-p
        (open-file res)
        (print 'cancelled))))

(defun prompt-for-file-working ()
  (capi:contain
   (make-instance
    'capi:push-button
    :text "Click Here"
    :callback-type :none
    :callback 'my-callback)))

(prompt-for-file-working)

```

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	<p>The <code>with-document-pages</code> evaluates <i>body</i> repeatedly, with <i>page-var</i> bound to the number of the page to print on each iteration. It is used to by applications providing Page on Demand printing.</p> <p>The <i>first-page</i> and <i>last-page</i> arguments are evaluated to yield the page numbers of the first and last pages in the document.</p> <p><code>with-document-pages</code> takes care of <i>first-page</i> and <i>last-page</i> when the user sets them in <code>print-dialog</code>, by evaluating <i>body</i> for the pages that are in the intersection of what user chose and the other arguments.</p>
See also	<p><code>print-dialog</code>  <code>with-page</code>  <code>with-print-job</code></p>

## with-external-metafile

Macro

Summary	Creates a metafile on disk using Graphics Ports operations.
Package	<code>capi</code>
Signature	<code>with-external-metafile (var &amp;key <i>pane bounds format pathname</i>) &amp;body <i>body</i> =&gt; nil</code>
Arguments	<p><i>var</i>            A variable.</p> <p><i>pane</i>           A graphics port, or <code>nil</code>.</p> <p><i>bounds</i>        A list of four integers. Can also be <code>nil</code> on Microsoft Windows.</p> <p><i>format</i>        One of <code>:enhanced</code> or <code>:windows</code>. Used only on Microsoft Windows.</p> <p><i>pathname</i>      A pathname or string.</p> <p><i>body</i>           Code containing Graphic Ports operations that draw to <i>var</i>.</p>

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 GTK+ or 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).
<code>%ratio%</code>	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):

<code>%width%</code>	The width in pixels of the pane.
<code>%height%</code>	The height in pixels of the pane.
<code>%min-width%</code>	The minimum width of the pane.
<code>%min-height%</code>	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.

`%scroll-vertical-page-size%`

The vertical scroll page size.

`%scroll-vertical-slug-size%`

The height of the scroll bar slug.

`%scroll-vertical-step-size%`

The vertical scroll step size.

`%scroll-x%` x coordinate of the current scroll position.

`%scroll-y%` y coordinate of the current scroll position

See also `convert-relative-position`  
`element`  
`get-horizontal-scroll-parameters`  
`get-vertical-scroll-parameters`  
`scroll`  
`set-horizontal-scroll-parameters`  
`set-vertical-scroll-parameters`

## with-internal-metafile

*Macro*

**Summary** Creates a metafile in memory using Graphics Ports operations.

**Package** `capi`

**Signature** `with-internal-metafile (var &key pane bounds format) &body body => metafile`

**Description** The macro `with-internal-metafile` creates a metafile 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-internal-metafile` is not implemented on GTK+ or X11/Motif.

`with-internal-metafile` behaves like `with-external-metafile` except that an object representing the metafile is returned, and no file is created on disk.

*metafile* must be freed after use, by calling `free-metafile`.

**Examples** There is an example in `examples/capi/graphics/metafile.lisp`.

See also `draw-metafile`  
`free-metafile`  
`with-external-metafile`

## with-output-to-printer

*Macro*

Summary Binds a stream variable and prints its output.

Package `capi`

Signature `with-output-to-printer (stream &key printer  
tab-spacing interactive jobname)  
&body body => result`

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 *result* The result of evaluating *body*.

Description The macro `with-output-to-printer` binds the variable *stream* to a stream object, and prints everything is that is written to it in the code of *body*.

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 `capi`

Signature `with-page-transform (x y width height) &body body`

Description     The `with-page-transform` macro evaluates *x*, *y*, *width* and *height* to define a rectangular region within the coordinate space of an output pane or printer port. Within *body* 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.

Any number of calls to `with-page-transform` 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.

See also         `get-printer-metrics`

## **with-print-job**

*Macro*

Summary         Creates a print job that prints to the specified printer.

Package         `capi`

Signature        `with-print-job (var &key pane jobname printer) &body body`

Description     The `with-print-job` macro creates a print job which prints to *printer*. If *printer* is not specified, the default printer is used. The macro binds *var* to a graphics port object, and printing is performed by using graphics port operations to draw the object.

If *pane* is specified 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 the `with-print-job` so all drawing operations draw to the printer instead of *pane*. This can be useful when implementing printing by modifying existing redisplay code that is written expecting an `output-pane`.

*jobname* is the name of the print job. The default value is `nil`, meaning that the name "Document" is used.

See also `printer-port-handle`  
`printer-port-supports-p`  
`set-printer-options`  
`with-document-pages`  
`with-page`  
`with-page-transform`

## **with-random-typeout** *Macro*

Summary      Binds a stream variable to a collector pane.

Package      `capi`

Signature     `with-random-typeout (stream-variable pane) &body body`

Description    The macro `with-random-typeout` binds the variable *stream-variable* to a collector pane stream associated with *pane* for the scope of the macro. The collector pane is automatically mapped and unmapped around the body. If the body exits normally, the typeout is not unmapped until the space bar is pressed or the mouse is clicked.

See also      `map-typeout`  
`unmap-typeout`  
`collector-pane`

## **wrap-text** *Function*

Summary      Wraps text for a given character width.

Package      `capi`

Signature	<code>wrap-text text width &amp;key start end =&gt; strings</code>	
Arguments	<code>text</code>	A string.
	<code>width</code>	A positive integer.
	<code>start, end</code>	Bounding index designators of <code>text</code> .
Values	<code>strings</code>	A list of strings.
Description	The function <code>wrap-text</code> takes a string <code>text</code> and returns a list of strings, each of which is no longer than <code>width</code> . Together the strings in <code>strings</code> contain all the non-whitespace characters of <code>text</code> between <code>start</code> and <code>end</code> and are suitable for displaying this text on multiple lines of length <code>width</code> .	
See also	<code>wrap-text-for-pane</code>	

## **wrap-text-for-pane**

*Function*

Summary	Wraps text for a given pane.	
Package	<code>capi</code>	
Signature	<code>wrap-text-for-pane pane text &amp;key external-width visible-width font start end =&gt; strings</code>	
Arguments	<code>text</code>	A string.
	<code>pane</code>	A displayed CAPI pane.
	<code>external-width</code>	An integer or <code>nil</code> .
	<code>visible-width</code>	An integer or <code>nil</code> .
	<code>font</code>	A font object.
	<code>start</code>	An integer.
	<code>end</code>	An integer or <code>nil</code> .

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-nil then text is wrapped to that width. Otherwise, if <i>external-width</i> is non-nil 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 nil, 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 nil.</p> <p>The <i>font</i> is used to perform the wrapping calculations. If it is nil (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	<code>wrap-text</code>	

## x-y-adjustable-layout

*Class*

Summary	<p>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).</p>	
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> .  <pre>(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)</pre>
See also	<code>pane-adjusted-position</code>



# 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 =&gt; 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</code> <i>transform dx dy</i>	
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> &amp;rest <i>font-attribute*</i> =&gt; <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> &amp;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 =&gt; 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 &amp;optional font =&gt; 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> . <b>Note:</b> 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>&amp;key cache force-new =&gt; 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 =&gt; 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 &amp;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

<i>pane</i>	A graphics port for a window.
<i>width</i>	An integer.
<i>height</i>	An integer.
<i>background</i>	A color designator.
<i>collect</i>	A boolean.
<i>relative</i>	A boolean.
<i>clear</i>	A list or <code>t</code> .

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 `t`, 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 <code>rgb-color-spec</code> should result in a pixel that is on in a 1-bit dithered bitmap. The current set of dithers is used in the decision.	
	<b>Note:</b> 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</i> <i>&amp;rest args &amp;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 &amp;rest args &amp;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><b>Note:</b> 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 &amp;rest args &amp;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 ( <i>x y</i> ). 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.	
Example	<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.

**Example**

```
(gp:draw-ellipse port 100 100 20 40)

(gp:draw-ellipse port 100 100 50 10
 :filled t
 :foreground :green)
```

**draw-image***Function*

<b>Summary</b>	Displays an image on a graphics port at a given position.	
<b>Package</b>	<code>graphics-ports</code>	
<b>Signature</b>	<code>draw-image port image to-x to-y &amp;rest args &amp;key from-x from-y to-width to-height from-width from-height global-alpha</code>	
<b>Arguments</b>	<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. Scaling is supported on GTK.

*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 or GTK. 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      `graphics-ports`

Signature     `draw-line port from-x from-y to-x to-y &rest args`

Arguments    *port*              A graphics port.

<i>from-x</i>	A real number.
<i>from-y</i>	A real number.
<i>to-x</i>	A real number.
<i>to-y</i>	A real number.
<i>args</i>	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

<i>port</i>	A graphics port.
<i>description</i>	A description sequence.
<i>args</i>	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</code> <i>port</i> <i>x</i> <i>y</i> &rest <i>args</i>	
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</i> <i>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</code> <i>port</i> <i>description</i> &rest <i>args</i>	
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`

<b>draw-string</b>	<i>Function</i>
Summary	Draws a string with the baseline positioned at a given point.
Package	<code>graphics-ports</code>
Signature	<code>draw-string port string x y &amp;rest args &amp;key start end block</code>
Arguments	<p><i>port</i>            A graphics port.</p> <p><i>string</i>          A string.</p> <p><i>x</i>                A real number.</p> <p><i>y</i>                A real number.</p> <p><i>start</i>           A real number.</p> <p><i>end</i>             A real number.</p> <p><i>block</i>           A boolean.</p> <p><i>args</i>            General graphics port drawing arguments.</p>
Description	<p>Draws the string with the baseline starting at (<i>x y</i>). 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 port's graphics state (see <code>make-graphics-state</code>) are all used. <i>start</i> and <i>end</i> specify which elements of the <i>string</i> to draw. <i>block</i>, if non-<code>nil</code>, causes each character to be drawn in a character cell filled with the port's graphics state <i>background</i>.</p> <p>By default, <i>start</i> is 0.</p> <p>You can draw with the system highlight by setting Graphics State slots <code>foreground :color_highlighttext</code> and <code>background :color_highlight</code>.</p> <p><b>Note:</b> The Graphics State slot <i>operation</i> is not supported for drawing text on Windows.</p>
See also	<code>make-graphics-state</code>

## 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 &amp;key event-func force shutdown =&gt; result</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><b>Note:</b> <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"><li>1. Load the GDI+ module <code>gdiplus.dll</code>, if it is not already loaded.</li></ol>	

2. If
  - a) GDI+ was already started by a previous call to `ensure-gdiplus`, 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 `(setf external-image-color-table)` replaces the color table in *external-image*. The color table specified by *replacement-color-table* must be the same length as the external image's original color table. It is a vector of RGB color-specifications.

**externalize-image***Function*

Summary Returns an external image containing color information from an image.

Package `graphics-ports`

Signature `externalize-image gp image &key maximum-colors important-colors &allow-other-keys => external-image`

Arguments *gp* A CAPI pane.  
*image* An image.  
*maximum-colors* An integer or `nil`. The default is `nil`.  
*important-colors* An integer or `nil`

Values *external-image* An external image.

Description The `externalize-image` function returns an `external-image` containing color information from *image*.

If *maximum-colors* is `nil` or if the screen has no palette, an `external-image` using all the colors in *image* is created.

If *maximum-colors* is an integer, the `external-image` containing image will be created using no more than that number of colors. If the image contains more than *maximum-colors* colors, the *maximum-colors* 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.

The value of *important-color* is recorded in the *external-image* 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.

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	<code>graphics-ports</code>	
Signature	<code>find-matching-fonts <i>pane</i> <i>fdesc</i> =&gt; <i>fonts</i></code>	
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 <i>font</i> =&gt; <i>fdesc</i></code>	
Arguments	<i>font</i>	A font.
Values	<i>fdesc</i>	A font description.

Description     The `font-description` function returns a font description object for *font*. Using this font description in a later call to `find-matching-fonts` or `find-best-font` on the original pane is expected to return a similar font.

See also         `convert-to-font-description`  
                  `make-font-description`

## 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.

	<i>font-attribute</i>	A font attribute.
Values	<i>value</i>	A font attribute value.
Description	The <code>font-description-attribute-value</code> function returns the value of <i>font-attribute</i> in <i>fdesc</i> , or <code>:wild</code> if <i>font-attribute</i> is not specified in <i>fdesc</i> .	

**font-fixed-width-p***Function*

Summary	Returns <code>t</code> if a specified font is of a fixed width.	
Package	<code>graphics-ports</code>	
Signature	<code>font-fixed-width-p port &amp;optional font =&gt; 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 <code>t</code> if the font associated with <i>port</i> , or the optionally specified <i>font</i> , is fixed width.  <b>Note:</b> <code>editor-pane</code> supports variable width fonts on Microsoft Windows, GTK+ and Motif.	

**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 =&gt; 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 <code>get-bounds</code> functions returns the four values <i>left</i> , <i>top</i> , <i>right</i> , <i>bottom</i> of the currently collected drawing extremes. The values can be used to get an image from the port.	
Example	<pre>(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)))</pre> <p>produces the following output:</p> <pre>72 112 285 255</pre>	
See also	<code>make-image-from-port</code>	

**get-character-extent***Function*

Summary	Returns the extent of a character in pixels.	
Package	<code>graphics-ports</code>	
Signature	<code>get-character-extent</code> <i>port character</i> &optional <i>font</i> => <i>left, top, right, bottom</i>	
Arguments	<i>port</i>	A CAPI pane.

	<i>character</i>	A character.
	<i>font</i>	A font.
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-character-extent</code> function returns the extent in pixels of the <i>character</i> in the font associated with <i>port</i> , or the <i>font</i> given.	

## get-char-ascent

*Function*

Summary	Returns the ascent of a character in pixels.	
Package	<code>graphics-ports</code>	
Signature	<code>get-char-ascent port character font =&gt; ascent</code>	
Arguments	<i>port</i>	A CAPI pane.
	<i>character</i>	A character.
	<i>font</i>	A font.
Values	<i>ascent</i>	An integer.
Description	The <code>get-character-ascent</code> function returns the <i>ascent</i> in pixels of the <i>character</i> in the font associated with <i>port</i> , or the <i>font</i> given.	

**get-char-descent***Function*

Summary	Returns the descent of a character in pixels.	
Package	<code>graphics-ports</code>	
Signature	<code>get-char-descent</code> <i>port character font</i> => <i>descent</i>	
Arguments	<i>port</i>	A CAPI pane.
	<i>character</i>	A character.
	<i>font</i>	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</code> <i>port character font</i> => <i>width</i>	
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 &amp;rest <i>points</i> =&gt; <i>left, top, right, bottom</i></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>nil</code> s.	

## get-font-ascent

*Function*

Summary	Returns the ascent of a font.	
Package	<code>graphics-ports</code>	
Signature	<code>get-font-ascent <i>port</i> &amp;optional <i>font</i> =&gt; <i>ascent</i></code>	
Arguments	<i>port</i>	A CAPI pane.
	<i>font</i>	A font.
Values	<i>ascent</i>	An integer.

Description     The `get-font-ascent` function returns the *ascent* in pixels of the font associated with *port*, or the *font* given.

## get-font-average-width

*Function*

Summary         Returns the average width of a font in pixels.

Package         `graphics-ports`

Signature       `get-font-average-width port &optional font => average-width`

Arguments       *port*             A CAPI pane.  
                   *font*             A font.

Values          *average-width*   An integer.

Description     The `get-font-average-width` function returns the *average-width* in pixels of the font associated with *port*, or the *font* given.

## get-font-descent

*Function*

Summary         Returns the descent in pixels of a font.

Package         `graphics-ports`

Signature       `get-font-descent port &optional font => descent`

Arguments       *port*             A CAPI pane.  
                   *font*             A font.

Values          *descent*         An integer.

Description     The `get-font-descent` function returns the *descent* in pixels of the font associated with *port*, or the *font* given.

## get-font-height

*Function*

Summary         Returns the height of a font.

Package         `graphics-ports`

Signature       `get-font-height port &optional font => height`

Arguments       *port*             A CAPI pane.

*font*             A font.

Values          *height*          An integer.

Description     The `get-font-height` function returns the *height* in pixels of the font associated with *port*, or the *font* given.

## get-font-width

*Function*

Summary         Returns the width of a font.

Package         `graphics-ports`

Signature       `get-font-width port &optional font => width`

Arguments       *port*             A graphics port.

*font*             A font.

Values          *width*          An integer.

Description     The `get-font-width` function returns the *width* in pixels of the font associated with *port*, or the *font* 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 <i>port</i> =&gt; <i>state</i></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> =&gt; <i>x y</i></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 `get-origin` values tell you by how much. The values are *not* needed when making images from the port's drawing.

```
Example (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)))
```

produces:

```
-15
0
```

## get-string-extent

*Function*

Summary Returns the extent in pixels of a string.

Package `graphics-ports`

Signature `get-string-extent` *port string* &optional *font*  
=> *left, top, right, bottom*

Arguments *port* A CAPI pane.

*string* A string.

Values *left* An integer.

*top* An integer.

*right* An integer.

*bottom* An integer.

Description The `get-string-extent` function returns the extent in pixels of the *string* in the font associated with *port*, or the *font* given.

**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`

## get-transform-scale

*Function*

Summary Returns the overall scaling factor of a transform.

Package `graphics-ports`

Signature `get-transform-scale transform => result`

Arguments *transform* A transform.

Values *result* A real number.

Description The `get-transform-scale` function returns a single number representing the overall scaling factor present in the *transform*.

## graphics-port-transform

*Function*

Summary Returns the transform object of a graphics port.

Package `graphics-ports`

Signature `graphics-port-transform port => transform`

Arguments *port* A graphics port.

Values *transform* A transform object.

Description     The `graphics-port-transform` function returns the transform object (a six-element list) associated with *port*.

## image

*Class*

Summary         An abstract image object. An image can be drawn via `draw-image`.

Package         `graphics-ports`

Accessors       `image-height`  
`image-width`

Description     The `image` class is the abstract image object class. An image can be drawn using `draw-image`.

`image-height` and `image-width` return the image size in pixels.

See also         `convert-external-image`  
`draw-image`  
`load-image`  
`make-image-from-port`  
`make-sub-image`  
`read-and-convert-external-image`

## image-access-pixel

*Function*

Summary         Gets and sets the pixels in an Image Access object.

Package         `graphics-ports`

Signature       `image-access-pixel` *image-access* *x* *y* => *color-rep*  
`(setf image-access-pixel)` *color-rep* *image-access* *x* *y* =>  
*color-rep*

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	<p>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>.</p> <p>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.</p> <p>The function <code>(setf image-access-pixel)</code> sets the value of the pixel at position <i>x</i>, <i>y</i> in the 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>	
Example	<p>See these example files@</p> <pre>examples/capi/graphics/image-access.lisp examples/capi/graphics/image-access-alpha.lisp</pre>	
See also	<pre>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</pre>	

## **image-access-pixels-from-bgra**

*Function*

Summary      Copies a vector of pixel values into an Image Access object.

Package	<code>graphics-ports</code>
Signature	<code>image-access-pixels-from-bgra</code> <i>image-access</i> <i>vector</i>
Arguments	<i>image-access</i> An Image Access object. <i>vector</i> A vector.
Description	<p>The function <code>image-access-pixels-from-bgra</code> copies all the pixels to the Image Access object <i>image-access</i> from the vector <i>vector</i>. <i>vector</i> 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 <i>vector</i> has element type <code>(unsigned-byte 8)</code>.</p> <p>An error is signalled if <i>vector</i> is not of the correct length for the Image Access object, that is <code>(* 4 <i>width</i> <i>height</i>)</code> where <i>width</i> and <i>height</i> represent the size of <i>image-access</i>.</p> <p><b>Note:</b> <code>image-access-pixels-to-bgra</code> must be called after this function (similarly to <code>(setf image-access-pixel)</code>). <i>image-access</i> must be an Image Access object returned by <code>make-image-access</code>.</p>
Example	See the file <code>examples/capi/graphics/image-access-bgra.lisp</code> .
See also	<code>image-access-pixel</code> <code>image-access-pixels-to-bgra</code>

## image-access-pixels-to-bgra

*Function*

Summary	Copies pixel values from an Image Access object into a vector.
Package	<code>graphics-ports</code>

Signature	<code>image-access-pixels-to-bgra</code> <i>image-access</i> <i>vector</i>
Arguments	<i>image-access</i> An Image Access object. <i>vector</i> A vector.
Description	<p>The function <code>image-access-pixels-to-bgra</code> copies all the pixels in the Image Access object <i>image-access</i> into the vector <i>vector</i> 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 <i>vector</i> has element type <code>(unsigned-byte 8)</code>.</p> <p>An error is signalled if <i>vector</i> is not of the correct length for the Image Access object, that is <math>(* 4 \textit{width} \textit{height})</math> where <i>width</i> and <i>height</i> represent the size of <i>image-access</i>.</p> <p><b>Note:</b> <code>image-access-pixels-from-bgra</code> must be called before this function (similarly to <code>image-access-pixel</code>).</p> <p><i>image-access</i> must be an Image Access object returned by <code>make-image-access</code>.</p>
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 <code>image</code> object, making them accessible via a corresponding Image Access object <code>image-access</code>.</p> <p><code>image-access</code> 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 <code>image-access</code>, 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	<p>See the file <code>examples/capi/graphics/image-access.lisp</code>.</p>
See also	<p><code>image-access-transfer-to-image</code>  <code>image-access-pixel</code>  <code>free-image-access</code>  <code>make-image-access</code></p>

## image-access-transfer-to-image

*Function*

Summary	Sets the pixel values in an <code>image</code> .
Package	<code>graphics-ports</code>
Signature	<code>image-access-transfer-to-image</code> <i>image-access</i>
Arguments	<i>image-access</i> An Image Access object
Description	<p>The function <code>image-access-transfer-to-image</code> sets the pixel values in an <code>image</code> object from the values in a corresponding Image Access object <code>image-access</code>.</p> <p><code>image-access</code> must be an Image Access object returned by <code>make-image-access</code>.</p>

Notionally `image-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 `graphics-ports`

Signature `image-freed-p image => bool`

Arguments `image` An image object.

Values `bool` A boolean.

Description The `image-freed-p` function returns `non-nil` if the image has been freed, and `nil` otherwise.

## **image-loader**

*Function*

Summary Returns the image load function.

Package `graphics-ports`

Signature `image-loader image-id &key image-translation-table => loader`

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	<code>register-image-load-function</code> <code>register-image-translation</code>	

## image-translation

*Function*

Summary	Returns the translation for an image registered in its image translation table.	
Package	<code>graphics-ports</code>	
Signature	<code>image-translation</code> <i>image-id</i> &key <i>image-translation-table</i> => <i>translation</i>	
Arguments	<i>image-id</i>	An image identifier.
	<i>image-translation-table</i>	An image translation table.
Values	<i>translation</i>	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         *order*                    An integer.

Description        The `initialize-dithers` function initializes dither objects up to the given *order* (size =  $2 \wedge order$ ). By default, order is 3.  
**Note:** dithers do not affect drawing or the antialiasing that occurs when drawing in Cocoa.

See also            `dither-color-spec`  
                       `make-dither`  
                       `with-dither`

## inset-rectangle

*Function (inline)*

Summary            Moves the corners of a rectangle inwards by a given amount.

Package            `graphics-ports`

Signature          `inset-rectangle rectangle dx dy &optional dx-right dy-bottom`

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> x y =&gt; <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	<p>The <code>inside-rectangle</code> function returns <code>t</code> if the point (<i>x y</i>) is inside <i>rectangle</i>. The <i>rectangle</i> is expected to be ordered; if the rectangle is specified by (<i>left right top 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.</p>	

**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</code> <i>object</i> <code>&amp;optional x y width height</code> <code>=&gt; result</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>&amp;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</code> <i>transform</i> <code>&amp;optional into</code> <code>=&gt; inverse</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 pane =&gt; fdescs</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 port id &amp;key width height =&gt; image</code>																		
Arguments	<p><i>port</i>            A graphics port or CAPI object.</p> <p><i>id</i>                A keyword, string or pathname.</p> <p><i>width</i>            The desired width in pixels, or <code>nil</code>.</p> <p><i>height</i>            The desired height in pixels, or <code>nil</code>.</p>																		
Values	<i>image</i> An <code>image</code> object.																		
Description	<p>The <code>load-icon-image</code> function loads an icon specified by <i>id</i> which should be either a keyword describing a standard icon, or a string or a pathname naming a Windows format icon (<code>.ico</code>) file. In this case, the first icon in the file is loaded.</p> <p>The following keyword values of <i>id</i> are recognized:</p> <table> <tr> <td><code>:sample</code></td> <td>A rectangle</td> </tr> <tr> <td><code>:hand</code></td> <td>A cross in a circle</td> </tr> <tr> <td><code>:ques</code></td> <td>A question mark in a bubble</td> </tr> <tr> <td><code>:bang</code></td> <td>An exclamation mark in a triangle</td> </tr> <tr> <td><code>:note</code></td> <td>An 'I' in a bubble</td> </tr> <tr> <td><code>:winlogo</code></td> <td>The Windows logo</td> </tr> <tr> <td><code>:warning</code></td> <td>Same as <code>:bang</code></td> </tr> <tr> <td><code>:error</code></td> <td>Same as <code>:hand</code></td> </tr> <tr> <td><code>:information</code></td> <td>Same as <code>:note</code></td> </tr> </table> <p><code>load-icon-image</code> returns an <code>image</code> object which can be drawn to <i>port</i> using <code>draw-image</code> and which must be freed using <code>free-image</code> when no longer needed.</p>	<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>
<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>																		

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 `graphics-ports`

Signature `load-image gp id &key cache type editable image-translation-table => image`

Arguments

- gp* A graphics port.
- id* An image identifier, a file, an `external-image`, or an `image`.
- cache* A boolean.
- type* A keyword, or `nil`.
- editable* One of the keywords `:with-alpha` and `:without-alpha`, or a boolean.
- image-translation-table*  
An image translation table.

Values *image* An image object.

Description The `load-image` function loads an image identified by *id* via the *image-translation-table* using the image load function registered with it. It returns an `image` object with the representation slot initialized. The *gp* 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 *id* 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 *image-translation-table* is `*default-image-translation-table*`.

*id* can be an `image`, which is just associated with the port *gp* and returned if it is a Plain Image or if *editable* is `nil`. Otherwise a new Plain Image object is returned, as described below.

*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:

`nil`                   The image is not editable.

- `:without-alpha` The image is editable, but does not have an alpha channel.
- `t` 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).
- `:with-alpha` 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 6.0 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 `make-dither` 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 `make-font-description` Returns a new font description object containing given font attributes.

Package `graphics-ports`

Signature	<code>make-font-description &amp;rest font-attribute* =&gt; fdesc</code>
Arguments	<i>font-attribute</i> A font attribute.
Values	<i>fdesc</i> A font description object.
Description	<p>The <code>make-font-description</code> function returns a new font description object containing the given font attributes. There is no error checking of the attributes at this point.</p> <p>The <code>:stock</code> attribute is handled specially: it is omitted from <i>fdesc</i>, unless it is the only attribute specified.</p>
See also	<code>augment-font-description</code> <code>convert-to-font-description</code> <code>find-best-font</code> <code>find-matching-fonts</code> <code>font-description</code> <code>merge-font-descriptions</code>

## make-graphics-state

*Function*

Summary	Creates a graphics state object.
Package	<code>graphics-ports</code>
Signature	<code>make-graphics-state &amp;key 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 =&gt; state</code>
Arguments	<i>transform</i> An object which determines the coordinate transformation applying to the graphics port. The default value is the unit transform which leaves the port coordinates unchanged from those used by the host window system — origin at top left, X

- increasing to the right and Y increasing down the screen. Allowed values are anything returned by the transform functions, described in section "Graphics state transforms" of the *LispWorks CAPI User Guide*.
- foreground* Determines the foreground color used in drawing functions. The value can be a pixel value, a color name symbol, a color name string or a color spec object. Using pixel values results in better performance. The default value is `:black`. The value `:color_highlighttext` is useful for drawing text with the system highlighting.
- background* Determines the background color used in drawing functions which use a stipple. Valid values are the same as for *foreground*. The default value is `:white`. The value `:color_highlight` is useful for drawing text with the system highlighting.
- operation* Determines the color combination used in the drawing primitives. Valid values are 0 to 15, being the same logical values as the *op* arg to the Common Lisp function `boolean`. The default value is `boolean-1`. The section "Graphics state operation" in the *LispWorks CAPI User Guide* shows how to use *operation*.
- stipple* A 1-bit pixmap ("bitmap") or `nil` (which is the default value). The bitmap is used in conjunction with the *fill-style* when drawing. Here, `nil` means that all pixels are drawn in the *foreground* color. A stipple is not transformed by the *transform* parameter. Its origin is assumed to coincide with the origin of the port. The *stipple* is tiled across the drawing. *stipple* is ignored if a *pattern* is given. If no

*fill-style* is given, or it is specified as `:solid`, when a *stipple* is given, then *fill-style* defaults to `:opaque-stippled`.

*fill-style* Determines how the drawing is done. The value should be one of `:solid`, `:opaque-stippled`, `:stippled` or `:tiled`. The default value `:solid` means that the *foreground* is used everywhere. `:opaque-stippled` means that the *stipple* bitmap is used with stipple 1s giving the *foreground* and 0s the *background*. `:stippled` means that the *stipple* bitmap is used with *foreground* where there are 1s and where there are 0s, no drawing is done. If you specify a stipple but no *fill-style*, or a *fill-style* of `:solid`, it defaults to `:opaque-stipple`.

*pattern* An image the same depth as the port, or `nil`. If non-`nil`, *pattern* is used as the source of color for drawing instead of the *foreground* and *background* parameters. A pattern is not transformed by the *transform* parameter. The *pattern* is tiled across the drawing. When *pattern* is specified, the *stipple* value is ignored. The default value of *pattern* is `nil`. See "Working with images" in the *LispWorks CAPI User Guide* for information on creating an image.

*thickness* A number (defaulting to 1) specifying the thickness of lines drawn. If *scale-thickness* is non-`nil`, the value *thickness* is in port (transformed) coordinates, otherwise *thickness* is in pixels.

*scale-thickness* A boolean, defaulting to `t` which means interpret the *thickness* parameter in transformed port coordinates. If *scale-thickness* is `nil`, *thickness* is interpreted in pixels.

<i>dashed</i>	A boolean, defaulting to <code>nil</code> . If <i>dashed</i> is <code>t</code> then lines are drawn as a dashed line using <i>dash</i> as the mark-space specifier.
<i>dash</i>	A list of two or more integer, or <code>nil</code> . A list of integers specifies the alternate mark and space sizes for dashed lines. These mark and space values are interpreted in pixels only. The default value of <i>dash</i> is <code>(4 4)</code> .
<i>line-end-style</i>	The value should be one of <code>:butt</code> , <code>:round</code> or <code>:projecting</code> and specifies how to draw the ends of lines. The default value is <code>:butt</code> .
<i>line-joint-style</i>	The value should be one of <code>:bevel</code> , <code>:miter</code> or <code>:round</code> and specifies how to draw the areas where the edges of polygons meet. The default value is <code>:miter</code> .
<i>mask</i>	The value should be <code>nil</code> (the default) or a list of the form <code>(x y width height)</code> , defining a rectangle inside which the drawing is done. The <i>mask</i> is not tiled. A <i>mask</i> is not transformed by the <i>transform</i> parameter.
<i>mask-x</i>	An integer specifying in window coordinates where in the port the X coordinate of the mask origin is to be considered to be. The default value is 0.
<i>mask-y</i>	An integer specifying in window coordinates where in the port the Y coordinate of the mask origin is to be considered to be. The default value is 0.
<i>font</i>	Either <code>nil</code> or a portable font description or font object to be used by the <code>draw-character</code> and <code>draw-string</code> functions. See "Portable font descriptions" in the <i>LispWorks CAPI User Guide</i> for details. The default value is <code>nil</code> .

Values `state` A graphics state object.

Description The `make-graphics-state` 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 `with-graphics-state` 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 "Setting the graphics state" in the *LispWorks CAPI User Guide* for examples.

Notes `operation` is not supported for drawing text on Microsoft Windows.

`stipple`, `fill-style`, `mask-x` and `mask-y` are supported only on X11/Motif.

`pattern` is supported only on Microsoft Windows and X11/Motif.

*operation* is not supported by Cocoa/Core Graphics so this slot or argument is ignored on Cocoa.

See also `set-graphics-state`

## make-image

*Function*

Summary 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	<code>graphics-ports</code>
Signature	<code>make-image-access port image =&gt; image-access</code>
Arguments	<i>port</i> A graphics port. <i>image</i> An <code>image</code> object.
Values	<i>image-access</i> An Image Access object.
Description	<p>The generic function <code>make-image-access</code> returns an Image Access object for the given <code>image</code> image.</p> <p><i>image</i> can be any <code>image</code> object returned by <code>make-image-from-port</code>. An <code>image</code> object returned by <code>load-image</code> is also suitable, but only if it is a Plain Image (see below).</p> <p><i>image-access</i> 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 <i>LispWorks CAPI User Guide</i>.</p> <p><b>Note:</b> on some platforms (currently Windows) not every <code>image</code> object is a Plain Image. If needed, forcibly create a Plain Image suitable for passing to <code>make-image-access</code> as described in <code>load-image</code>.</p> <p><b>Note:</b> ensure that you eventually discard <i>image-access</i>, using <code>free-image-access</code>.</p>
Example	See the file <code>examples/capi/graphics/image-access.lisp</code> .
See also	<code>free-image-access</code> <code>image-access-transfer-from-image</code> <code>image-access-transfer-to-image</code>

`image-access-pixel`  
`load-image`  
`make-image`

## make-image-from-port

*Function*

Summary	Makes an image out of a specified rectangle of a graphics port's contents.	
Package	<code>graphics-ports</code>	
Signature	<code>make-image-from-port</code> <i>port</i> &optional <i>x y width height</i> => <i>image</i>	
Arguments	<i>port</i>	A graphics port.
	<i>x</i>	An integer.
	<i>y</i>	An integer.
	<i>width</i>	An integer.
	<i>height</i>	An integer.
Values	<i>image</i>	An image.
Description	<p>The <code>make-image-from-port</code> function makes an <code>image</code> out of the specified rectangle of the port's contents. The default is the whole port, but a region can be specified using <i>x</i>, <i>y</i>, <i>width</i>, and <i>height</i>. The default value of <i>x</i> and <i>y</i> is 0.</p> <p>Normally the image is freed automatically, when <i>port</i> 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.</p>	
See also	<code>externalize-image</code>	

## make-sub-image

*Function*

Summary	Makes a new image from part of an image.	
Package	<code>graphics-ports</code>	
Signature	<code>make-sub-image port image &amp;optional x y width height =&gt; sub-image</code>	
Arguments	<i>port</i>	A graphics port.
	<i>image</i>	An image.
	<i>x</i>	An integer.
	<i>y</i>	An integer.
	<i>width</i>	An integer.
	<i>height</i>	An integer.
Values	<i>sub-image</i>	An image.
Description	<p>The function <code>make-sub-image</code> makes a new <code>image</code> object from the rectangular region of the supplied <i>image</i> specified by <i>x</i>, <i>y</i>, <i>width</i> and <i>height</i>.</p> <p>The default values of <i>x</i> and <i>y</i> are 0.</p> <p>The default value of <i>width</i> is the <i>width</i> of <i>image</i>.</p> <p>The default value of <i>height</i> is the <i>height</i> of <i>image</i>.</p>	
See also	<code>image</code>	

## 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</code>	&optional <i>a b c d e f</i> => <i>transform</i>
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	The <code>make-transform</code> function returns a new transform object initialized according to the optional args. The default args make the unit transform.	
	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	
	<pre> a b 0 c d 0 e f 1 </pre>	
	for generalized two dimensional points of the form ( <i>x y</i> 1).	

## 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</code>	<i>fdesc1 fdesc2</i> => <i>fdesc</i>
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</code> <i>rectangle dx dy</i>	
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>	



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 <code>pixmap-port</code> 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	<code>graphics-ports</code>	
Signature	<code>port-string-width</code> <i>port string</i> => <i>width</i>	
Arguments	<i>port</i>	A graphics port.
	<i>string</i>	A string.
Values	<i>width</i>	An integer.
Description	<p>The <code>port-string-width</code> function returns the <i>width</i> in pixels of <i>string</i> when drawn to <i>port</i>. The font used is the one currently in the port's graphics state.</p> <p><b>Note:</b> To compute the horizontal extents of each successive character in a string for a given port or font, use <code>compute-char-extents</code>.</p>	
See also	<code>compute-char-extents</code>	

## port-width

*Function*

Summary	Returns the pixel width of a port.	
Package	<code>graphics-ports</code>	
Signature	<code>port-width</code> <i>port</i> => <i>width</i>	
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 <i>transform1 transform2</i></code>	
Arguments	<i>transform1</i>	A transform.
	<i>transform2</i>	A transform.
Description	<p>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>.</p> <pre>transform1 = transform1 . transform2</pre>	

**premultiply-transforms***Function*

Summary	Premultiplies two transforms.	
Package	<code>graphics-ports</code>	
Signature	<code>premultiply-transforms <i>transform1 transform2</i></code>	
Arguments	<i>transform1</i>	A transform.
	<i>transform2</i>	A transform.
Description	<p>The <code>premultiply-transforms</code> function premultiplies 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</p>	

*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	<code>graphics-ports</code>
Signature	<code>read-and-convert-external-image <i>gp file</i> &amp;key <i>transparent-color-index</i> =&gt; <i>image, external-image</i></code>
Arguments	<i>gp</i> A CAPI pane. <i>file</i> A pathname designator. <i>transparent-color-index</i> An integer or <code>nil</code> .
Values	<i>image</i> An image. <i>external-image</i> An external image.
Description	Returns an image converted from an external image read from <i>file</i> . The external image is returned as a second value. <i>transparent-color-index</i> is interpreted as described for <code>read-external-image</code> .
See also	<code>convert-external-image</code> <code>external-image</code> <code>read-external-image</code>

<b>read-external-image</b>		<i>Function</i>
Summary	Returns an external image read from a file.	
Package	<code>graphics-ports</code>	
Signature	<code>read-external-image file &amp;key transparent-color-index type =&gt; image</code>	
Arguments	<i>file</i>	A pathname designator.
	<i>transparent-color-index</i>	An integer or <code>nil</code> .
	<i>type</i>	A keyword, or <code>nil</code> .
Values	<i>image</i>	An external image.
Description	<p>The <code>read-external-image</code> function returns an external image read from <i>file</i>.</p> <p><i>transparent-color-index</i> specifies the index of the transparent color in the color map. <i>transparent-color-index</i> works only for images with a color map, that is, those with 256 colors or less. The default value is <code>nil</code>, meaning that there is no transparent color.</p> <p><i>type</i> tells <code>read-external-image</code> that the image is in a particular graphics format. Currently the only recognised value is <code>:bmp</code>, which means the image is read as a Bitmap. Other values of <i>type</i> cause <code>read-external-image</code> to read the image according to the file type of <i>file</i>. <code>"bmp"</code> or <code>"dib"</code> 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 <i>LispWorks CAPI User Guide</i> for details. The default value of <i>type</i> is <code>nil</code>.</p>	
Example	To see the effect of <i>transparent-color-index</i> , edit <code>examples/capi/graphics/images.lisp</code> .	

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 `Setup.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	<code>graphics-ports</code>	
Signature	<code>rectangle-bind ((<i>a b c d</i>) <i>rectangle</i>) &amp;body <i>body</i> =&gt; <i>result</i></code>	
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 <code>rectangle-height</code> macro returns the difference between the <i>bottom</i> and <i>top</i> elements of <i>rectangle</i> . <i>rectangle</i> is a list of numbers ( <i>left top right bottom</i> ).	

## rectangle-left

Macro

Summary	Gets and set the <i>left</i> element of a rectangle.	
Package	<code>graphics-ports</code>	
Signature	<code>rectangle-left <i>rectangle</i> =&gt; <i>left</i></code>	
Signature	<code>(setf rectangle-left) <i>left rectangle</i> =&gt; <i>left</i></code>	
Arguments	<i>rectangle</i>	A rectangle.
Values	<i>left</i>	A real number.
Description	The <code>rectangle-left</code> macro returns and via <code>setf</code> sets the <i>left</i> element of <i>rectangle</i> . <i>rectangle</i> is a list of numbers ( <i>left top right bottom</i> ).	

## rectangle-right

Macro

Summary	Gets aand sets the <i>right</i> element of a rectangle.	
Package	<code>graphics-ports</code>	
Signature	<code>rectangle-right <i>rectangle</i> =&gt; <i>right</i></code>	
Signature	<code>(setf rectangle-right) <i>right rectangle</i> =&gt; <i>right</i></code>	
Arguments	<i>rectangle</i>	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</code> <i>rectangle</i> => <i>top</i>	
Signature	<code>(setf rectangle-top)</code> <i>top</i> <i>rectangle</i> => <i>top</i>	
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</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>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> =&gt; <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 ((<i>x y width height</i>) <i>rectangle</i>) &amp;body <i>body</i> =&gt; <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 <i>image-id image-load-function</i> &amp;key <i>image-translation-table</i></code>	
Arguments	<i>image-id</i>	An image identifier or a list of image identifiers.

*image-load-function*

A function.

*image-translation-table*

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. <i>image-translation-table</i> An image translation table. <i>image-load-fn</i> An image loading function.

Description	The <code>register-image-translation</code> function registers <i>image-id</i> and <i>image-load-fn</i> with the <i>translation</i> in the <i>image-translation-table</i> . When <code>load-image</code> is called with second argument <i>image-id</i> , the <i>image-load-fn</i> is called with <i>translation</i> as its second argument. If <i>image-load-fn</i> is <code>nil</code> , the image translation table's default image loader is used; this converts an external image object or file to an image. If <i>translation</i> is <code>nil</code> the identifier is deregistered. Returns the <i>image-id</i> and the <i>image-load-fn</i> . 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> <code>reset-image-translation-table</code>

## reset-image-translation-table

*Function*

Summary	Clears the image translation table hash tables.
Package	<code>graphics-ports</code>
Signature	<code>reset-image-translation-table &amp;key <i>image-translation-table</i></code>
Arguments	<i>image-translation-table</i> An image translation table.
Description	The <code>reset-image-translation-table</code> function clears the image translation table hash tables and set the default <i>image-load-fn</i> to <code>read-and-convert-external-image</code> . 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>read-and-convert-external-image</code> <code>register-image-translation</code>

## separation

*Function*

Summary	Returns the distance between two points.
Package	<code>graphics-ports</code>
Signature	<code>separation x1 y1 x2 y2 =&gt; dist</code>
Arguments	<i>x1</i> An integer. <i>y1</i> An integer. <i>x2</i> An integer. <i>y2</i> An integer.
Values	<i>dist</i> A real number.
Description	The <code>separation</code> function returns the distance between points ( <i>x1 y1</i> ) and ( <i>x2 y2</i> ).

## set-default-image-load-function

*Function*

Summary	Sets the default image load function of an image translation table.
Package	<code>graphics-ports</code>
Signature	<code>set-default-image-load-function <i>image-load-function</i> &amp;key <i>image-translation-table</i></code>
Arguments	<i>image-load-function</i> An image load function. <i>image-translation-table</i> An image translation function.



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	<code>graphics-ports</code>
Signature	<code>set-graphics-state</code> <i>port</i> &rest <i>args</i> <i>&amp;key transform foreground background</i> <i>operation stipple pattern fill-style thickness</i> <i>scale-thickness dashed dash line-end-style</i> <i>line-joint-style mask mask-x mask-y font</i>
Arguments	<i>port</i> A graphics port.
Description	This directly alters the graphics state of <i>port</i> according to the values of the keyword arguments <i>args</i> . Unspecified keywords leave the associated slots unchanged.  See <code>make-graphics-state</code> for valid values for <i>args</i> .
See also	<code>make-graphics-state</code> <code>with-graphics-state</code>

## transform

*Type*

Summary	The transform type, defined for transform objects.
Package	<code>graphics-ports</code>
Description	The <code>transform</code> 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	<code>graphics-ports</code>	
Signature	<code>transform-area transform x y width height =&gt; rectangle</code>	
Arguments	<i>transform</i>	A transform.
	<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>rectangle</i>	A rectangle.
Description	The <code>transform-area</code> 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	<code>graphics-ports</code>	
Signature	<code>transform-distance transform dx dy =&gt; dx2, dy2</code>	
Arguments	<i>transform</i>	A transform.
	<i>dx</i>	A real number.
	<i>dy</i>	A real number.
Values	<i>dx2</i>	A real number.

*dy2*                    A real number.

Description        The `transform-distance` function transforms the distance  $(dx\ dy)$  by the rotation and scale in the *transform*. The translation in the transform is ignored. Transformed  $(dx\ dy)$  is returned as two values.

## **transform-distances**

*Function*

Summary            Transforms a list of alternating distance vectors by a given transform.

Package            `graphics-ports`

Signature          `transform-distances transform distances => result`

Arguments         *transform*            A transform.  
*distances*            A list of pairs of real numbers.

Values             *result*                A list of pairs of real numbers.

Description        The `transform-distances` function transforms a list of alternating  $(dx\ dy)$  pairs in *distances* by the *transform*. Transformed values are returned as a new list.

## **transform-is-rotated**

*Function*

Summary            Returns `t` if a given transform contains a rotation.

Package            `graphics-ports`

Signature          `transform-is-rotated transform => bool`

Arguments         *transform*            A transform.

Values	<i>bool</i>	A boolean.
Description	The <code>transform-is-rotated</code> function returns <code>t</code> if <i>transform</i> 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 =&gt; xnew ynew</code>	
Arguments	<i>transform</i>	A transform.
	<i>x</i>	A real number.
	<i>y</i>	A real number.
Values	<i>xnew</i>	A real number.
	<i>ynew</i>	A real number.
Description	The <code>transform-point</code> function transforms the point ( <i>x y</i> ) by multiplying it by <i>transform</i> . The transformed ( <i>x y</i> ) 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 transform points &amp;optional into =&gt; 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>transform-points</code> function transforms a list of alternating $(x\ y)$ 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 =&gt; 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.
	<i>bottom2</i>	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            *rectangle*        A rectangle.

Description     The `union-rectangle` macro modifies the *rectangle* to be the union of *rectangle* and *(left top right bottom)*.

### **\*unit-transform\***

*Variable*

Summary         The list `(1 0 0 1 0 0)`.

Package         `graphics-ports`

Signature        `*unit-transform*`

Description     The `*unit-transform*` 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 `t` if a given transform is a unit transform.

Package         `graphics-ports`

Signature        `unit-transform-p transform => bool`

Arguments        *transform*         A transform.

Values            *bool*                A boolean.

Description     The `unit-transform-p` returns `t` if *transform* 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>) &amp;body <i>body</i> =&gt; <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> =&gt; <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 =&gt; 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 <i>transform</i> x y =&gt; 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 <i>transform</i> <i>points</i> &amp;optional <i>into</i> =&gt; <i>result</i></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</code> <i>object</i> <code>&amp;optional x y width height</code> => <i>result</i>	
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>&amp;optional</code> arguments.</p> <p>The <i>result</i> is <code>non-nil</code> if the function succeeds and <code>nil</code> if it fails (doing nothing).</p> <p><b>Note:</b> 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>) &amp;body <i>body</i> =&gt; <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><b>Note:</b> 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> &amp;key) &amp;body <i>body</i> =&gt; <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> <p style="text-align: center;"><i>mask-x + x</i> to <i>mask-x + x + width</i></p> <p>and whose Y coordinate is in the range</p> <p style="text-align: center;"><i>mask-y + y</i> to <i>mask-y + y + height</i></p>	

## 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) &amp;body body =&gt; 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 =&gt; 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 `with-graphics-state` macro binds the graphics state values for the specified port to the values specified in the *args* list. The keyword arguments *args* correspond to the slots in the graphics state, as described in `set-graphics-state`. See `make-graphics-state` for valid values for *args*.

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

```
make-graphics-state
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>) &amp;body <i>body</i> =&gt; <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>) &amp;body <i>body</i> =&gt; <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) &amp;body body =&gt; 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>) &amp;body <i>body</i> =&gt; <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) &amp;body body =&gt; 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

<i>external-image</i>	An external-image.
<i>file</i>	A file.
<i>if-exists</i>	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> =&gt; <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 key-word 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*

<b>Summary</b>	Returns colors and color aliases containing a given string.
<b>Package</b>	<code>color</code>
<b>Signature</b>	<code>apropos-color-names <i>substring</i> =&gt; <i>list</i></code>
<b>Arguments</b>	<i>substring</i> A string.
<b>Values</b>	<i>list</i> A list of symbols.
<b>Description</b>	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      `color-red color-spec => color-component`  
                  `color-green color-spec => color-component`  
                  `color-blue color-spec => color-component`  
                  `color-hue color-spec => color-component`  
                  `color-saturation color-spec => color-component`  
                  `color-value color-spec => color-component`

Arguments      `color-spec`      A color specification.

Values          `color-component`      A color component from the appropriate color model.

Description    If `color-spec` is not from the appropriate color model (`:rgb` in the case of `color-red`, `color-green` and `color-blue`, and `:hsv` in the case of `color-hue`, `color-saturation` and `color-value`) then the component is calculated.

Example        `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 >`

See also        `make-hsv`  
                  `make-rgb`  
                  `make-gray`  
                  `color-model`  
                  `color-level`

### **\*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 &gt; (color:make-gray 0.66667s0) #(:GRAY 0.66667S0)  COLOR 3 &gt; (color-level *) 0.66667S0  COLOR 4 &gt;</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 &amp;optional tolerance =&gt; 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-&lt;command&gt;</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 &amp;key errorp =&gt; 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 &amp;optional if-exists =&gt; 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 &gt; (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	<code>color</code>
Signature	<code>define-color-models <i>model-descriptors</i>=&gt; <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> =&gt; &lt;no values&gt;</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> =&gt; <i>result</i></code> <code>ensure-hsv <i>color-spec</i> =&gt; <i>result</i></code> <code>ensure-gray <i>color-spec</i> =&gt; <i>result</i></code> <code>ensure-model-color <i>color-spec model</i> =&gt; <i>result</i></code> <code>ensure-color <i>color-spec match-color-spec</i> =&gt; <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 &amp;optional sort =&gt; 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 =&gt; 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 &gt; (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 &gt; (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 &gt; (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 &gt; (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> =&gt; &lt;no values&gt;</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	<code>color</code>	
Signature	<code>make-hsv <i>hue saturation value</i> &amp;optional <i>alpha</i> =&gt; <i>color-spec</i></code>	
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	Return a <i>color-spec</i> in the <code>:hsv</code> model with components <i>hue</i> , <i>saturation</i> and <i>value</i> .	
	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.	
	<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.	
Example	<pre>COLOR 27 &gt; (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** *color-spec* 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 297
- Accelerators 226, 297
- accepts-focus-p generic function 4
- :accepts-focus-p initarg 53, 145
- accessor functions
  - application-interface-application-menu 46
  - application-interface-dock-menu 46
  - application-interface-message-callback 46
  - button-alternate-callback 425
  - button-armed-image 16
  - button-cancel-p 16
  - button-default-p 16
  - button-disabled-image 16
  - button-enabled 16
  - button-image 16
  - button-press-callback 425
  - button-selected 16
  - button-selected-disabled-image 16
  - button-selected-image 16
  - callbacks-action-callback 27
  - callbacks-callback-type 27
  - callbacks-extend-callback 27
  - callbacks-retract-callback 27
  - callbacks-selection-callback 27
  - capi-object-name 31
  - capi-object-plist 31
  - choice-initial-focus-item 35
  - choice-interaction 35
  - choice-selection 35
  - cocoa-view-pane-init-function 48
  - cocoa-view-pane-view-class 48
  - collection-items 53
  - collection-items-count-function 53
  - collection-items-get-function 53
  - collection-items-map-function 53
  - collection-print-function 53
  - collection-test-function 53
  - collector-pane-stream 58
  - display-pane-text 99
  - docking-layout-controller 104
  - docking-layout-divider-p 104
  - docking-layout-docking-test-function 104
  - docking-layout-items 104
  - docking-layout-orientation 104
  - document-frame-container 109
  - drawn-pinboard-object-display-callback 119
  - editor-pane-buffer 141
  - editor-pane-change-callback 132
  - editor-pane-enabled 132
  - editor-pane-fixed-fill 132
  - editor-pane-line-wrap-face 132
  - editor-pane-line-wrap-marker 132
  - editor-pane-text 132
  - editor-pane-wrap-style 132
  - element-interface 146
  - element-parent 146
  - element-widget-name 146
  - filled 152, 435
  - filtering-layout-matches-text 159

- filtering-layout-state 159
- form-title-adjust 168
- form-title-gap 168
- form-vertical-adjust 168
- form-vertical-gap 168
- graph-edge-from 176
- graph-edge-to 176
- graph-node-height 177
- graph-node-in-edges 177
- graph-node-out-edges 177
- graph-node-width 177
- graph-node-x 177
- graph-node-y 177
- graph-object-element 178
- graph-object-object 178
- graph-pane-layout-function 179
- graph-pane-roots 179
- help-key 53, 146, 298, 555
- image-height 649
- image-pinboard-object-image 196
- image-width 649
- interactive-pane-stream 201
- interactive-pane-top-level-function 201
- interface-activate-callback 207
- interface-confirm-destroy-function 207
- interface-create-callback 207
- interface-default-toolbar-states 207
- interface-destroy-callback 207
- interface-geometry-change-callback 207
- interface-help-callback 207
- interface-iconify-callback 207, 215
- interface-menu-bar-items 207
- interface-message-area 207, 215
- interface-override-cursor 207
- interface-pointer-documentation-enabled 207
- interface-title 207
- interface-toolbar-items 207
- interface-toolbar-states 207
- interface-tooltips-enabled 207
- interface-window-styles 207
- item-collection 238
- item-data 238
- item-print-function 238
- item-selected 238
- item-text 238
- labelled-line-text-foreground 241
- layout-description 242
- layout-ratios 60, 454
- layout-x-adjust 603
- layout-x-gap 189
- layout-x-ratios 189
- layout-y-adjust 603
- layout-y-gap 189
- layout-y-ratios 189
- list-panel-right-click-selection-behavior 246
- list-view-auto-reset-column-widths 258
- list-view-columns 258
- list-view-image-function 258
- list-view-state-image-function 258
- list-view-subitem-function 258
- list-view-subitem-print-functions 258
- list-view-view 258
- menu-image-function 290
- menu-items 290
- menu-object-enabled 303
- menu-popup-callback 303
- menu-title 545
- menu-title-function 545
- ole-control-component-pane 326
- option-pane-enabled 338
- option-pane-enabled-positions 338
- option-pane-image-function 338
- option-pane-popup-callback 338
- option-pane-separator-item 338
- option-pane-visible-items-count 338
- output-pane-create-callback 342
- output-pane-destroy-callback 342
- output-pane-display-callback 342
- output-pane-focus-callback 342
- output-pane-graphics-options 342
- output-pane-input-model 342
- output-pane-resize-callback 342
- output-pane-scroll-callback 342
- pane-layout 21, 207
- password-pane-overwrite-character 364
- pinboard-object-activep 370
- pinboard-object-graphics-args 370
- pinboard-object-pinboard 370
- popup-menu-button-menu 388
- popup-menu-button-menu-function 388
- range-callback 433

range-end 433  
 range-orientation 433  
 range-slug-end 433  
 range-slug-start 433  
 range-start 433  
 rich-text-pane-change-callback 444  
 rich-text-pane-limit 444  
 rich-text-pane-text 444  
 screen-depth 456  
 screen-height 456  
 screen-height-in-millimeters 456  
 screen-interfaces 108, 456  
 screen-number 456  
 screen-width 456  
 screen-width-in-millimeters 456  
 scroll-bar-line-size 462  
 scroll-bar-page-size 462  
 shell-pane-command 494  
 simple-pane-background 499  
 simple-pane-cursor 499  
 simple-pane-drag-callback 499  
 simple-pane-drop-callback 499  
 simple-pane-enabled 159, 499, 561  
 simple-pane-font 499  
 simple-pane-foreground 499  
 simple-pane-horizontal-scroll 499  
 simple-pane-scroll-callback 499  
 simple-pane-vertical-scroll 499  
 simple-pane-visible-border 499  
 slider-show-value-p 510  
 slider-start-point 510  
 switchable-layout-combine-child-  
   constraints 517  
 switchable-layout-visible-child  
   517  
 tab-layout-combine-child-con-  
   straints 519  
 tab-layout-visible-child-function  
   519  
 text-input-pane-before-change-  
   callback 534  
 text-input-pane-buttons-enabled  
   525  
 text-input-pane-callback 525  
 text-input-pane-caret-position 525  
 text-input-pane-change-callback  
   525  
 text-input-pane-completion-func-  
   tion 525  
 text-input-pane-confirm-change-  
   function 525  
 text-input-pane-editing-callback  
   525  
 text-input-pane-enabled 525  
 text-input-pane-max-characters 525  
 text-input-pane-navigation-call-  
   back 525  
 text-input-pane-text 525  
 text-input-range-callback 542  
 text-input-range-callback-type 542  
 text-input-range-end 542  
 text-input-range-start 542  
 text-input-range-value 542  
 text-input-range-wraps-p 542  
 titled-object-message 547  
 titled-object-message-font 215, 547  
 titled-object-title 547  
 titled-object-title-font 547  
 title-pane-text 544  
 toolbar-button-dropdown-menu 555  
 toolbar-button-dropdown-menu-  
   function 555  
 toolbar-button-dropdown-menu-kind  
   555  
 toolbar-button-image 555  
 toolbar-button-popup-interface 555  
 toolbar-button-selected-image 555  
 toolbar-flat-p 552  
 toolbar-object-enabled-function  
   561  
 top-level-interface-external-bor-  
   der 207  
 top-level-interface-transparency  
   207  
 tree-view-action-callback-expand-  
   p 572  
 tree-view-checkbox-change-call-  
   back 572  
 tree-view-checkbox-child-function  
   572  
 tree-view-checkbox-initial-status  
   572  
 tree-view-checkbox-next-map 572  
 tree-view-checkbox-parent-func-  
   tion 572  
 tree-view-checkbox-status 572  
 tree-view-children-function 572  
 tree-view-expandp-function 572  
 tree-view-has-root-line 572  
 tree-view-image-function 572  
 tree-view-leaf-node-p-function 572  
 tree-view-retain-expanded-nodes  
   572  
 tree-view-right-click-extended-

- match 572
  - tree-view-roots 572
  - tree-view-state-image-function 572
- :action-callback initarg 27
- :action-callback-expand-p initarg 570
- :activate-callback initarg 205
- activate-pane function 5
- :activep initarg 369, 432
- active-pane-copy function 6
- active-pane-copy-p function 6
- active-pane-cut function 6
- active-pane-cut-p function 6
- active-pane-deselect-all function 6
- active-pane-deselect-all-p function 6
- active-pane-paste function 6
- active-pane-paste-p function 6
- active-pane-select-all function 6
- active-pane-select-all-p function 6
- active-pane-undo function 6
- active-pane-undo-p function 6
- ActiveX 332
- :adjust initarg 60, 454
- :adjust item in :buttons initarg 533
- :after-input-callback initarg 131
- :alternate-callback initarg 425
- :alternative initarg 297
- analyze-external-image function 605
- append-items generic function 7
- Application menu 45
- application-interface-application-menu accessor function 46
- application-interface-dock-menu accessor function 46
- application-interface-message-callback accessor function 46
- :application-menu initarg 46
- apply-in-pane-process function 8
- apply-rotation function 606
- apply-scale function 606
- apply-translation function 607
- apropos-color-alias-names function 717
- apropos-color-names function 718
- apropos-color-spec-names function 719
- :armed-image initarg 16
- :armed-images initarg 20
- arrow-pinboard-object class 9
- attach-interface-for-callback func-

- tion 11
- attach-simple-sink function 12
- attach-sink function 13
- augment-font-description function 607
- :auto-menus initarg 204
- :auto-reset-column-widths initarg 257, 310

## B

- background graphics state parameter 666
- :background initarg 497
- beep-pane function 14
- :before-change-callback initarg 534
- :before-input-callback initarg 131
- :best-height initarg 204
- :best-width initarg 204
- :best-x initarg 204
- :best-y initarg 204
- :browse-file item in :buttons initarg 531
- :buffer-modes initarg 131
- :buffer-name initarg 58, 131
- built-in scrolling 174
- button class 15
- button-alternate-callback accessor function 425
- button-armed-image accessor function 16
- button-cancel-p accessor function 16
- :button-class initarg 20
- button-default-p accessor function 16
- button-disabled-image accessor function 16
- button-enabled accessor function 16
- :button-height initarg 552
- button-image accessor function 16
- button-panel class 20
- button-press-callback accessor function 425
- :buttons initarg 525
- button-selected accessor function 16
- button-selected-disabled-image accessor function 16
- button-selected-image accessor function 16
- :button-width initarg 552

## C

- calculate-constraints generic function 25
- calculate-layout generic function 26
- :callback initarg 15, 17, 159, 303, 432, 462, 524, 542, 554

- :callback-data-function initarg 303
- :callback-object initarg 159
- callbacks 27
  - for button panels 20
  - for buttons 17
  - passing different variables 12
- callbacks class 27
- :callbacks initarg 20, 552, 560
- callbacks-action-callback accessor function 27
- callbacks-callback-type accessor function 27
- callbacks-extend-callback accessor function 27
- callbacks-retract-callback accessor function 27
- callbacks-selection-callback accessor function 27
- :callback-type initarg 27, 519, 524, 542
- call-editor generic function 29
- :cancel item in :buttons initarg 531
- cancel-button image identifier 534
- :cancel-button initarg 21
- :cancel-function item in :buttons initarg 532
- :cancel-p initarg 15
- CAPI process 93
- capi-object class 30
- capi-object-name accessor function 31
- capi-object-plist accessor function 31
- capi-object-property function 31
- :caret-position initarg 524
- :change-callback initarg 131, 159, 444, 525
- :change-callback-type initarg 524
- :character-format initarg 444
- :checkbox-change-callback initarg 571
- :checkbox-child-function initarg 571
- :checkbox-initial-status initarg 571
- :checkbox-next-map initarg 571
- :checkbox-parent-function initarg 571
- :checkbox-status initarg 571
- check-button class 32
- check-button-panel class 33
- %child% geometry slot 594
- :child initarg 508
- :children-function initarg 178, 569
- choice class 34
- choice-initial-focus-item accessor function 35
- choice-interaction accessor function 35
- choice-selected-item generic function 38
- choice-selected-item-p function 39
- choice-selected-items generic function 40
- choice-selection accessor function 35
- choice-update-item function 41
- class options
  - :coclass 88
  - :definition 80
  - :interfaces 88
  - :layouts 80
  - :menu-bar 80
  - :menus 80
  - :panes 80
  - :source-interfaces 88
- classes
  - arrow-pinboard-object 9
  - button 15
  - button-panel 20
  - callbacks 27
  - capi-object 30
  - check-button 32
  - check-button-panel 33
  - choice 34
  - cocoa-default-application-interface 45
  - cocoa-view-pane 48
  - collection 52
  - collector-pane 58
  - color-screen 59
  - column-layout 60
  - display-pane 98
  - docking-layout 103
  - document-container 108
  - document-frame 109
  - double-headed-arrow-pinboard-object 112
  - double-list-panel 113
  - drawn-pinboard-object 119
  - echo-area-pane 129
  - editor-pane 131
  - element 144
  - ellipse 152
  - expandable-item-pinboard-object 157
  - extended-selection-tree-view 158
  - external-image 632
  - filtering-layout 158
  - foreign-owned-interface 167
  - form-layout 168
  - graph-edge 176
  - graph-node 176

- graph-object 177
- graph-pane 178
- grid-layout 188
- image 649
- image-list 195
- image-pinboard-object 196
- image-set 197
- interactive-pane 201
- interface 204
- item 237
- item-pinboard-object 239
- labelled-arrow-pinboard-object 240
- labelled-line-pinboard-object 240
- layout 241
- line-pinboard-object 243
- listener-pane 261
- list-panel 245
- list-view 256
- menu 290
- menu-component 294
- menu-item 297
- menu-object 303
- message-pane 307
- mono-screen 309
- multi-column-list-panel 310
- multi-line-text-input-pane 314
- non-focus-list-interface 315
- ole-control-component 326
- ole-control-doc 328
- ole-control-frame 329
- ole-control-pane 332
- ole-control-pane-simple-sink 335
- option-pane 337
- output-pane 340
- password-pane 364
- pinboard-layout 366
- pinboard-object 369
- pixmap-port 677
- popup-menu-button 388
- progress-bar 401
- push-button 425
- push-button-panel 426
- radio-button 429
- radio-button-panel 430
- range-pane 432
- rectangle 434
- rich-text-pane 444
- right-angle-line-pinboard-object 452
- row-layout 453
- screen 455
- scroll-bar 462
- shell-pane 493
- simple-layout 496
- simple-network-pane 496
- simple-pane 497
- simple-pinboard-layout 508
- slider 510
- sorted-object 512
- static-layout 514
- switchable-layout 516
- tab-layout 518
- text-input-choice 522
- text-input-pane 523
- text-input-range 542
- titled-menu-object 544
- titled-object 546
- titled-pinboard-object 549
- title-pane 543
- toolbar 552
- toolbar-button 554
- toolbar-component 559
- toolbar-object 561
- tracking-pinboard-layout 567
- tree-view 569
- x-y-adjustable-layout 602
- clear-external-image-conversions function 608
- clear-graphics-port function 609
- clear-graphics-port-state function 609
- clear-rectangle function 610
- clipboard function 42
- clipboard-empty function 44
- clone generic function 45
- :close-callback initarg 332
- :coclass class option 88
- Cocoa Event Loop process 93
- cocoa-default-application-interface class 45
- cocoa-view-pane class 48
- cocoa-view-pane-init-function accessor function 48
- cocoa-view-pane-view function 50
- cocoa-view-pane-view-class accessor function 48
- collect-interfaces generic function 51
- collection class 52
- :collection initarg 238
- collection-find-next-string generic function 55
- collection-find-string generic function 56

- collection-items accessor function 53
- collection-items-count-function  
accessor function 53
- collection-items-get-function acces-  
sor function 53
- collection-items-map-function acces-  
sor function 53
- collection-last-search generic func-  
tion 57
- collection-print-function accessor  
function 53
- collection-search generic function 57
- collection-test-function accessor  
function 53
- collector-pane class 58
- collector-pane-stream accessor function  
58
- color-<component> function 720
- \*color-database\* variable 721
- :color-function initag 246
- color-level function 720, 722
- color-model function 723
- colors= function 723
- color-screen class 59
- :column initag 188
- :column-function initag 310
- column-layout class 60
- column-layout-divider 61
- :columns initag 310
- :combine-child-constraints initag  
516, 519
- :command initag 493
- command table 341
- complete-button image identifier 534
- :complete-do-action initag 525
- :completion item in :buttons initag 531
- :completion-function initag 524
- component-name function 63
- :component-name initag 332, 335
- compress-external-image function 610
- compute-char-extents function 611
- comtab 341
- Confirm Before Exiting 64, 470
- :confirm-change-function initag 525
- :confirm-destroy-function initag 204
- confirmer-pane function 65
- confirm-quit function 63
- confirm-yes-or-no function 65
- contain function 66
- container 109
- container special slot 109
- continuation function, dialog

- creating 589
- using 95, 352, 381, 391, 403, 405, 408,  
410, 412, 413, 415, 418, 420, 421,  
423, 424
- :controller initag 103
- convert-color function 724
- convert-external-image function 611
- convert-relative-position function 67
- convert-to-font-description function  
612
- convert-to-screen function 68
- copy-external-image function 613
- copy-pixels function 613
- copy-transform function 614
- count-collection-items generic func-  
tion 72
- :create-callback initag 204, 326, 341
- create-pixmap-port function 615
- current-dialog-handle function 72
- current-document generic function 74
- current-pointer-position function 74
- current-popup function 75
- current-printer function 76
- :cursor initag 498

## D

- dash graphics state parameter 668
- dashed graphics state parameter 668
- :data initag 238
- :default initag 242
- :default-button initag 71
- \*default-editor-pane-line-wrap-  
marker\* variable 76
- :default-image-set initag 552, 560
- \*default-image-translation-table\*  
variable 616, 656
- default-library function 77
- :default-p initag 15
- :default-toolbar-states initag 206
- define-color-alias function 725
- define-color-models macro 726
- define-command macro 77
- define-font-alias function 616
- define-interface macro 79
- define-layout macro 86
- define-menu macro 88
- define-ole-control-component macro  
87
- :definition class option 80
- delete-color-translation function 727
- :depth initag 456
- :description initag 242, 519

- destroy button
  - removal 212
- destroy generic function 89
- :destroy-callback initarg 204, 326, 341
- destroy-pixmap-port function 617
- detach-simple-sink function 90
- detach-sink function 91
- dialog continuation function
  - creating 589
  - using 95, 352, 381, 391, 403, 405, 408, 410, 412, 413, 415, 418, 420, 421, 423, 424
- dialogs
  - aborting 2
- :directories-only initarg 524
- :disabled-image initarg 15
- :disabled-images initarg 20
- display function 92
- :display-callback initarg 119, 341
- display-dialog function 94
- display-errors macro 97
- display-message function 97
- display-message-for-pane function 98
- display-message-on-screen function 98
- display-pane class 98
- display-pane-text accessor function 99
- display-popup-menu function 100
- display-replacable-dialog function 101
- :display-state initarg 206
- display-tooltip generic function 102
- dither-color-spec function 617
- :divider-p initarg 103
- :dividerp initarg 552
- Dock menu 45
- :docking-callback initarg 103
- docking-layout class 103
- docking-layout-controller accessor function 104
- docking-layout-divider-p accessor function 104
- docking-layout-docking-test-function accessor function 104
- docking-layout-items accessor function 104
- docking-layout-orientation accessor function 104
- docking-layout-pane-docked-p function 107
- docking-layout-pane-visible-p function 107
- :docking-test-function initarg 103
- :dock-menu initarg 46
- document-container class 108
- document-frame class 109
- document-frame-container accessor function 109
- double-headed-arrow-pinboard-object class 112
- :double-head-predicate initarg 112
- double-list-panel class 113
- Drag and drop
  - coordinates 127
  - dragging 115
  - dropping 498
  - effect 122, 125
  - formats 127, 473
  - object 126
- :drag-callback initarg 498
- drag-pane-object function 115
- draw-arc function 618
- draw-arcs function 619
- draw-character function 620
- draw-circle function 620
- draw-ellipse function 621
- draw-image function 622
- draw-line function 623
- draw-lines function 624
- draw-metafile function 117
- draw-metafile-to-image function 118
- drawn-pinboard-object class 119
- drawn-pinboard-object-display-callback accessor function 119
- draw-pinboard-object generic function 120
- draw-pinboard-object-highlighted generic function 121
- draw-pinboard-object-unhighlighted generic function 121
- draw-point function 625
- draw-points function 625
- draw-polygon function 626
- draw-polygons function 627
- draw-rectangle function 628
- draw-rectangles function 629
- draw-string function 630
- :draw-with-buffer initarg 342
- :drop-callback initarg 498
- :dropdown-menu initarg 555
- :dropdown-menu-function initarg 555
- :dropdown-menu-kind initarg 555

- drop-object-allows-drop-effect-p function 122
- drop-object-collection-index function 123
- drop-object-collection-item function 124
- drop-object-drop-effect function 125
- drop-object-get-object function 126
- drop-object-pane-x function 127
- drop-object-pane-y function 127
- drop-object-provides-format function 127

## E

- :echo-area initarg 132
- \*echo-area-cursor-inactive-style\* variable 128
- echo-area-pane class 129
- :edge-pinboard-class initarg 179
- :editing-callback initarg 525
- \*editor-cursor-active-style\* variable 130
- \*editor-cursor-color\* variable 129
- \*editor-cursor-drag-style\* variable 130
- \*editor-cursor-inactive-style\* variable 130
- editor-pane class 131
- editor-pane-blink-rate generic function 140
- editor-pane-buffer accessor function 141
- editor-pane-change-callback accessor function 132
- editor-pane-enabled accessor function 132
- editor-pane-fixed-fill accessor function 132
- editor-pane-line-wrap-face accessor function 132
- editor-pane-line-wrap-marker accessor function 132
- editor-pane-native-blink-rate function 142
- editor-pane-selected-text function 142
- editor-pane-selected-text-p generic function 143
- editor-pane-stream function 143
- editor-pane-text accessor function 132
- editor-pane-wrap-style accessor function 132
- editor-window generic function 144

- element class 144
- element-container function 151
- element-interface accessor function 146
- element-interface-for-callback function 151
- element-parent accessor function 146
- element-screen function 152
- element-widget-name accessor function 146
- ellipse class 152
- :enabled initarg 15, 131, 159, 303, 337, 497, 524, 561
- :enabled-function initarg 303, 561
- :enabled-function-for-dialog initarg 298
- :enabled-positions initarg 338
- :enabled-slot initarg 303
- :enable-pointer-documentation initarg 205
- :enable-tooltips initarg 205
- :end-x initarg 244
- :end-y initarg 244
- ensure-<command> function 728
- ensure-area-visible generic function 153
- ensure-gdiplus function 631
- ensure-interface-screen function 153
- Escape key 379
- event handler
  - key strokes 342
  - mouse click 342
  - mouse gestures 342
  - mouse move 342
- execute-with-interface function 154
- execute-with-interface-if-alive function 155
- exit-confirmer function 156
- exit-dialog function 156
- expandable-item-pinboard-object class 157
  - :expandp-function initarg 570
  - :extend-callback initarg 27
- extended-selection-tree-view class 158
- :external-border initarg 206
- external-image class 632
- external-image-color-table function 633
- externalize-image function 634
- :external-max-height initarg 145, 370
- :external-max-width initarg 145, 370
- :external-min-height initarg 145, 369
- :external-min-width initarg 145, 369

## F

- :file-completion initarg 524
- :filename initarg 444
- filled accessor function 152, 435
- :filled initarg 152, 435
- fill-style graphics state parameter 667
- :filter initarg 246
- :filter-automatic-p initarg 246
- :filter-callback initarg 246
- :filter-change-callback-p initarg 246
- :filter-help-string initarg 246
- filtering-layout class 158
- filtering-layout-matches-text accessor function 159
- filtering-layout-match-object-and-exclude-p function 163
- filtering-layout-state accessor function 159
- :filter-matches-title initarg 246
- :filter-short-menu-text initarg 246
- find-best-font function 635
- find-graph-edge generic function 163
- find-graph-node generic function 164
- finding panes
  - interfaces 80
- find-interface generic function 165
- find-matching-fonts function 636
- find-pane 80
- find-string-in-collection generic function 166
- :fit-size-to-children initarg 514
- :fixed-fill initarg 132
- :flatp initarg 552
- focus
  - keyboard input on Cocoa 213
  - mouse events on Cocoa 213
  - moving to a new pane 5
  - setting to a pane 356, 482
- :focus-callback initarg 341
- font graphics state parameter 668
- :font initarg 497
- font-description function 636
- font-description-attributes function 637
- font-description-attribute-value function 637
- font-fixed-width-p function 638
- force-screen-update function 166
- force-update-all-screens function 167
- foreground graphics state parameter 666
- :foreground initarg 497
- foreign-owned-interface class 167
- form-layout class 168
- form-title-adjust accessor function 168
- form-title-gap accessor function 168
- form-vertical-adjust accessor function 168
- form-vertical-gap accessor function 168
- frame 547
- free-image function 638
- free-image-access function 639
- free-metafile function 169
- free-sound function 169
- :from initarg 176
- functions
  - abort-callback 1
  - abort-dialog 2
  - abort-exit-confirmer 3
  - activate-pane 5
  - active-pane-copy 6
  - active-pane-copy-p 6
  - active-pane-cut 6
  - active-pane-cut-p 6
  - active-pane-deselect-all 6
  - active-pane-deselect-all-p 6
  - active-pane-paste 6
  - active-pane-paste-p 6
  - active-pane-select-all 6
  - active-pane-select-all-p 6
  - active-pane-undo 6
  - active-pane-undo-p 6
  - analyze-external-image 605
  - apply-in-pane-process 8
  - apply-rotation 606
  - apply-scale 606
  - apply-translation 607
  - apropos-color-alias-names 717
  - apropos-color-names 718
  - apropos-color-spec-names 719
  - attach-interface-for-callback 11
  - attach-simple-sink 12
  - attach-sink 13
  - augment-font-description 607
  - beep-pane 14
  - capi-object-property 31
  - choice-selected-item-p 39
  - choice-update-item 41
  - clear-external-image-conversions 608
  - clear-graphics-port 609
  - clear-graphics-port-state 609

clear-rectangle 610  
clipboard 42  
clipboard-empty 44  
cocoa-view-pane-view 50  
color-<component> 720  
color-level 720, 722  
color-model 723  
colors= 723  
component-name 63  
compress-external-image 610  
compute-char-extents 611  
confirmer-pane 65  
confirm-quit 63  
confirm-yes-or-no 65  
contain 66  
convert-color 724  
convert-external-image 611  
convert-relative-position 67  
convert-to-font-description 612  
convert-to-screen 68  
copy-external-image 613  
copy-pixels 613  
copy-transform 614  
create-pixmap-port 615  
current-dialog-handle 72  
current-pointer-position 74  
current-popup 75  
current-printer 76  
default-library 77  
define-color-alias 725  
define-font-alias 616  
delete-color-translation 727  
destroy-pixmap-port 617  
detach-simple-sink 90  
detach-sink 91  
display 92  
display-dialog 94  
display-message 97  
display-message-for-pane 98  
display-message-on-screen 98  
display-popup-menu 100  
display-replacable-dialog 101  
dither-color-spec 617  
docking-layout-pane-docked-p 107  
docking-layout-pane-visible-p 107  
drag-pane-object 115  
draw-arc 618  
draw-arcs 619  
draw-character 620  
draw-circle 620  
draw-ellipse 621  
draw-image 622  
draw-line 623  
draw-lines 624  
draw-metatile 117  
draw-metatile-to-image 118  
draw-point 625  
draw-points 625  
draw-polygon 626  
draw-polygons 627  
draw-rectangle 628  
draw-rectangles 629  
draw-string 630  
drop-object-allows-drop-effect-p  
122  
drop-object-collection-index 123  
drop-object-collection-item 124  
drop-object-drop-effect 125  
drop-object-get-object 126  
drop-object-pane-x 127  
drop-object-pane-y 127  
drop-object-provides-format 127  
editor-pane-native-blink-rate 142  
editor-pane-selected-text 142  
editor-pane-stream 143  
element-container 151  
element-interface-for-callback 151  
element-screen 152  
ensure-<command> 728  
ensure-gdiplus 631  
ensure-interface-screen 153  
execute-with-interface 154  
execute-with-interface-if-alive  
155  
exit-confirmer 156  
exit-dialog 156  
external-image-color-table 633  
externalize-image 634  
filtering-layout-match-object-  
and-exclude-p 163  
find-best-font 635  
find-matching-fonts 636  
font-description 636  
font-description-attributes 637  
font-description-attribute-value  
637  
font-fixed-width-p 638  
force-screen-update 166  
force-update-all-screens 167  
free-image 638  
free-image-access 639  
free-metatile 169  
free-sound 169  
get-all-color-names 729

- get-bounds 639
- get-character-extent 640
- get-char-ascent 641
- get-char-descent 642
- get-char-width 642
- get-color-alias-translation 730
- get-color-spec 731
- get-constraints 170
- get-enclosing-rectangle 643
- get-font-ascent 643
- get-font-average-width 644
- get-font-descent 644
- get-font-height 645
- get-font-width 645
- get-graphics-state 646
- get-origin 646
- get-page-area 172
- get-printer-metrics 173
- get-scroll-position 174
- get-string-extent 647
- get-transform-scale 648
- graphics-port-transform 648
- graphics-state-background 669
- graphics-state-dash 669
- graphics-state-dashed 669
- graphics-state-fill-style 669
- graphics-state-font 669
- graphics-state-foreground 669
- graphics-state-line-end-style 669
- graphics-state-line-joint-style 669
- graphics-state-mask 669
- graphics-state-mask-x 669
- graphics-state-mask-y 669
- graphics-state-operation 669
- graphics-state-pattern 669
- graphics-state-scale-thickness 669
- graphics-state-stipple 669
- graphics-state-thickness 669
- graphics-state-transform 669
- graph-pane-edges 185
- graph-pane-nodes 186
- graph-pane-object-at-position 186
- hide-interface 193
- image-access-pixel 649
- image-access-pixels-from-bgra 650
- image-access-pixels-to-bgra 651
- image-access-transfer-from-  
image 652
- image-access-transfer-to-image 653
- image-freed-p 654
- image-loader 654
- image-translation 655
- initialize-dithers 656
- inset-rectangle 656
- inside-rectangle 657
- installed-libraries 200
- install-postscript-printer 198
- interface-customize-toolbar 220
- interface-display-title 222
- interface-iconified-p 225
- interface-preserving-state-p 230
- interface-toolbar-state 232
- interface-visible-p 234
- invalidate-pane-constraints 236
- invalidate-rectangle 658
- invert-transform 658
- invoke-command 236
- invoke-untranslated-command 237
- line-pinboard-object-coordinates 244
- list-all-font-names 659
- listener-pane-insert-value 262
- list-panel-items-and-filter 254
- load-color-database 732
- load-cursor 263
- load-icon-image 659
- load-image 661
- load-sound 266
- lower-interface 268
- make-dither 664
- make-docking-layout-controller 270
- make-font-description 664
- make-foreign-owned-interface 270
- make-general-image-set 272
- make-graphics-state 665
- make-gray 733
- make-hsv 734
- make-icon-resource-image-set 273
- make-image 670
- make-image-from-port 672
- make-image-locator 274
- make-menu-for-pane 274
- make-resource-image-set 277
- make-rgb 735
- make-scaled-general-image-set 278
- make-scaled-image-set 279
- make-sorting-description 280
- make-sub-image 673
- make-transform 673

map-typeout 289  
merge-font-descriptions 674  
modify-editor-pane-buffer 308  
non-focus-list-add-filter 316  
non-focus-list-remove-filter 316  
non-focus-list-toggle-enable-filter 315  
non-focus-list-toggle-filter 316  
offset-rectangle 675  
ole-control-add-verbs 324  
ole-control-close-object 325  
ole-control-i-dispatch 330  
ole-control-insert-object 331  
ole-control-ole-object 331  
ole-control-pane-frame 335  
ole-control-user-component 336  
ordered-rectangle-union 675  
page-setup-dialog 351  
pane-close-display 355  
pane-descendant-child-with-focus 356  
pane-supports-menus-with-images 363  
pixblt 676  
play-sound 365  
popup-confirmer 377  
port-height 678  
port-string-height 678  
port-string-width 679  
port-width 679  
postmultiply-transforms 680  
premultiply-transforms 680  
print-dialog 390  
print-editor-buffer 392  
printer-configuration-dialog 395  
printer-metrics-device-height 396  
printer-metrics-device-width 396  
printer-metrics-dpi-x 396  
printer-metrics-dpi-y 396  
printer-metrics-height 396  
printer-metrics-left-margin 396  
printer-metrics-max-height 396  
printer-metrics-max-width 396  
printer-metrics-min-left-margin 397  
printer-metrics-min-top-margin 397  
printer-metrics-paper-height 397  
printer-metrics-paper-width 397  
printer-metrics-top-margin 396  
printer-metrics-width 396  
printer-port-handle 397  
printer-port-supports-p 398  
print-file 392  
print-rich-text-pane 393  
print-text 394  
process-pending-messages 400  
prompt-for-color 401  
prompt-for-confirmation 402  
prompt-for-directory 404  
prompt-for-file 406  
prompt-for-files 409  
prompt-for-font 410  
prompt-for-form 411  
prompt-for-forms 413  
prompt-for-integer 414  
prompt-for-items-from-list 415  
prompt-for-number 416  
prompt-for-string 417  
prompt-for-symbol 419  
prompt-for-value 421  
prompt-with-list 422  
prompt-with-list-non-focus 319  
prompt-with-message 424  
quit-interface 428  
raise-interface 431  
range-set-sizes 433  
read-and-convert-external-image 681  
read-color-db 736  
read-external-image 682  
read-sound-file 434  
rectangle-union 686  
redisplay-menu-bar 436  
redraw-pinboard-layout 437  
redraw-pinboard-object 437  
register-image-load-function 688  
register-image-translation 689  
remove-capi-object-property 438  
replace-dialog 440  
reset-image-translation-table 690  
reuse-interfaces-p 443  
rich-text-pane-character-format 446  
rich-text-pane-operation 447  
rich-text-pane-paragraph-format 451  
rich-text-version 452  
screen-active-interface 457  
screen-active-p 457  
screen-internal-geometry 458  
screen-logical-resolution 458  
screens 459  
selection 465  
selection-empty 466

separation 691  
set-application-interface 467  
set-clipboard 468  
set-confirm-quit-flag 470  
set-default-editor-pane-blink-rate 470  
set-default-image-load-function 691  
set-default-interface-prefix-suffix 471  
set-drop-object-supported-formats 473  
set-editor-parenthesis-colors 474  
set-geometric-hint 475  
set-graphics-port-coordinates 692  
set-graphics-state 693  
set-hint-table 475  
set-interactive-break-gestures 477  
set-object-automatic-resize 478  
set-printer-metrics 488  
set-printer-options 489  
set-rich-text-pane-character-format 482  
set-rich-text-pane-paragraph-format 485  
set-selection 487  
set-text-input-pane-selection 490  
show-interface 495  
show-pane 495  
simple-pane-handle 505  
simple-print-port 509  
sort-object-items-by 511  
start-gc-monitor 513  
stop-gc-monitor 515  
stop-sound 516  
tab-layout-panes 521  
tab-layout-visible-child 522  
text-input-pane-complete-text 536  
text-input-pane-copy 537  
text-input-pane-cut 538  
text-input-pane-delete 538  
text-input-pane-in-place-complete 539  
text-input-pane-paste 539  
text-input-pane-selected-text 540  
text-input-pane-selection 540

text-input-pane-selection-p 541  
transform-area 694  
transform-distance 694  
transform-distances 695  
transform-is-rotated 695  
transform-point 696  
transform-points 696  
transform-rect 697  
tree-view-ensure-visible 578  
tree-view-item-checkbox-status 579  
tree-view-item-children-checkbox-status 580  
unconvert-color 737  
undefine-font-alias 698  
uninstall-postscript-printer 582  
unit-transform-p 699  
unmap-typeout 583  
untransform-distance 700  
untransform-distances 701  
untransform-point 701  
untransform-points 702  
update-all-interface-titles 584  
update-pinboard-object 585  
update-screen-interface-titles 585  
update-toolbar 586  
validate-rectangle 703  
wrap-text 600  
wrap-text-for-pane 601  
write-external-image 714

## G

:gap initarg 60, 454  
generic functions  
  accepts-focus-p 4  
  append-items 7  
  calculate-constraints 25  
  calculate-layout 26  
  call-editor 29  
  choice-selected-item 38  
  choice-selected-items 40  
  clone 45  
  collect-interfaces 51  
  collection-find-next-string 55  
  collection-find-string 56  
  collection-last-search 57  
  collection-search 57  
  count-collection-items 72  
  current-document 74  
  destroy 89  
  display-tooltip 102  
  draw-pinboard-object 120  
  draw-pinboard-object-highlighted

121  
 draw-pinboard-object-unhighlighted 121  
 editor-pane-blink-rate 140  
 editor-pane-selected-text-p 143  
 editor-window 144  
 ensure-area-visible 153  
 find-graph-edge 163  
 find-graph-node 164  
 find-interface 165  
 find-string-in-collection 166  
 get-collection-item 170  
 get-horizontal-scroll-parameters 171  
 get-vertical-scroll-parameters 175  
 graph-node-children 177  
 graph-pane-add-graph-node 182  
 graph-pane-delete-object 183  
 graph-pane-delete-objects 183  
 graph-pane-delete-selected-objects 184  
 graph-pane-direction 184  
 graph-pane-select-graph-nodes 187  
 graph-pane-update-moved-objects 187  
 hide-pane 193  
 highlight-pinboard-object 194  
 interactive-pane-execute-command 203  
 interface-display 221  
 interface-editor-pane 223  
 interface-extend-title 223  
 interface-geometry 224  
 interface-keys-style 225  
 interface-match-p 228  
 interface-menu-groups 229  
 interface-preserve-state 230  
 interface-reuse-p 231  
 interpret-description 235  
 itemp 239  
 list-panel-enabled 252  
 list-panel-filter-state 253  
 list-panel-unfiltered-items 255  
 locate-interface 267  
 make-container 269  
 make-image-access 671  
 make-pane-popup-menu 275  
 manipulate-pinboard 282  
 map-collection-items 285  
 map-pane-children 285  
 map-pane-descendant-children 288  
 merge-menu-bars 306  
 move-line 309  
 non-focus-maybe-capture-gesture 316  
 non-focus-terminate 318  
 non-focus-update 319  
 over-pinboard-object-p 351  
 pane-adjusted-offset 352  
 pane-adjusted-position 353  
 pane-got-focus 356  
 pane-has-focus-p 357  
 pane-initial-focus 357  
 pane-interface-copy-object 359  
 pane-interface-copy-p 359  
 pane-interface-cut-object 359  
 pane-interface-cut-p 359  
 pane-interface-deselect-all 359  
 pane-interface-deselect-all-p 359  
 pane-interface-paste-object 359  
 pane-interface-paste-p 359  
 pane-interface-select-all 359  
 pane-interface-select-all-p 359  
 pane-interface-undo 359  
 pane-interface-undo-p 359  
 pane-popup-menu-items 360  
 pane-string 362  
 parse-layout-descriptor 364  
 pinboard-object-at-position 372  
 pinboard-object-graphics-arg 373  
 pinboard-object-overlap-p 374  
 pinboard-pane-position 375  
 pinboard-pane-size 376  
 print-capi-button 388  
 print-collection-item 389  
 redisplay-collection-item 435  
 redisplay-interface 435  
 reinitialize-interface 438  
 remove-items 439  
 replace-items 441  
 report-active-component-failure 442  
 scroll 460  
 scroll-if-not-visible-p 463  
 search-for-item 465  
 set-button-panel-enabled-items 468  
 set-horizontal-scroll-parameters 476  
 set-pane-focus 482  
 set-scroll-position 461  
 set-scroll-range 476, 493  
 set-top-level-interface-geometry 491  
 set-vertical-scroll-parameters 492

simple-pane-visible-height 506  
 simple-pane-visible-size 506  
 simple-pane-visible-width 507  
 sorted-object-sort-by 512  
 switchable-layout-switchable-children 518  
 top-level-interface 561  
 top-level-interface-display-state 562  
 top-level-interface-geometry 563  
 top-level-interface-geometry-key 564  
 top-level-interface-p 566  
 top-level-interface-save-geometry-p 566  
 tree-view-expanded-p 578  
 tree-view-update-an-item 580  
 tree-view-update-item 581  
 unhighlight-pinboard-object 582  
 update-interface-title 584  
 geometry slots  
   %child% 594  
   %height% 594  
   %max-height% 595  
   %max-width% 595  
   %min-height% 594  
   %min-width% 594  
   %object% 594  
   %ratio% 594  
   %scroll-height% 595  
   %scroll-horizontal-page-size% 595  
   %scroll-horizontal-slug-size% 595  
   %scroll-horizontal-step-size% 595  
   %scroll-start-x% 595  
   %scroll-start-y% 595  
   %scroll-vertical-page-size% 595  
   %scroll-vertical-slug-size% 595  
   %scroll-vertical-step-size% 595  
   %scroll-width% 595  
   %scroll-x% 595  
   %scroll-y% 595  
   %width% 594  
   %x% 594  
   %y% 594  
 :geometry-change-callback initarg 205  
 :gesture-callbacks initarg 525  
 get pane  
   interface 80  
   get-all-color-names function 729  
   get-bounds function 639  
   get-character-extent function 640  
   get-char-ascent function 641  
   get-char-descent function 642  
   get-char-width function 642  
   get-collection-item generic function 170  
   get-color-alias-translation function 730  
   get-color-spec function 731  
   get-constraints function 170  
   get-enclosing-rectangle function 643  
   get-font-ascent function 643  
   get-font-average-width function 644  
   get-font-descent function 644  
   get-font-height function 645  
   get-font-width function 645  
   get-graphics-state function 646  
   get-horizontal-scroll-parameters generic function 171  
   get-origin function 646  
   get-page-area function 172  
   get-pane 80  
   get-printer-metrics function 173  
   get-scroll-position function 174  
   get-string-extent function 647  
   get-transform-scale function 648  
   get-vertical-scroll-parameters generic function 175  
   graph-edge class 176  
   graph-edge-from accessor function 176  
   graph-edge-to accessor function 176  
   :graphics-args initarg 369  
   :graphics-options initarg 341  
   graphics-port-transform function 648  
   graphics-state-background function 669  
   graphics-state-dash function 669  
   graphics-state-dashed function 669  
   graphics-state-fill-style function 669  
   graphics-state-font function 669  
   graphics-state-foreground function 669  
   graphics-state-line-end-style function 669  
   graphics-state-line-joint-style function 669  
   graphics-state-mask function 669  
   graphics-state-mask-x function 669  
   graphics-state-mask-y function 669  
   graphics-state-operation function 669  
   graphics-state-pattern function 669  
   graphics-state-scale-thickness func-

- tion 669
- graphics-state-stipple function 669
- graphics-state-thickness function 669
- graphics-state-transform function 669
- graph-node class 176
- graph-node-children generic function 177
- graph-node-height accessor function 177
- graph-node-in-edges accessor function 177
- graph-node-out-edges accessor function 177
- graph-node-width accessor function 177
- graph-node-x accessor function 177
- graph-node-y accessor function 177
- graph-object class 177
- graph-object-element accessor function 178
- graph-object-object accessor function 178
- graph-pane class 178
- graph-pane-add-graph-node generic function 182
- graph-pane-delete-object generic function 183
- graph-pane-delete-objects generic function 183
- graph-pane-delete-selected-objects generic function 184
- graph-pane-direction generic function 184
- graph-pane-edges function 185
- graph-pane-layout-function accessor function 179
- graph-pane-nodes function 186
- graph-pane-object-at-position function 186
- graph-pane-roots accessor function 179
- graph-pane-select-graph-nodes generic function 187
- graph-pane-update-moved-objects generic function 187
- grid-layout class 188
- groupbox 547

## H

- :has-root-line initag 571
- :has-title-column-p initag 188
- :head initag 9
- :head-breadth initag 9
- :head-direction initag 9
- :header-args initag 310

- :head-graphics-args initag 9
- :head-length initag 9
- %height% geometry slot 594
- :height initag 456
- help
  - context help 213
  - help-callback 209
- :help item in :buttons initag 532
- :help-callback initag 205, 206
- help-key accessor function 53, 146, 298, 555
- :help-key initag 53, 145, 297, 555
- :help-keys initag 21
- :help-string initag 159
- hide-interface function 193
- hide-pane generic function 193
- highlight-pinboard-object generic function 194
- :highlight-style initag 366
- :horizontal-scroll initag 463, 498
- HWND 72, 505

## I

- :iconify-callback initag 205
- :ignore-file-suffices initag 524
- image class 649
- image identifiers
  - cancel-button 534
  - complete-button 534
  - ok-button 533
- :image initag 15, 196, 554
- image-access-pixel function 649
- image-access-pixels-from-bgra function 650
- image-access-pixels-to-bgra function 651
- image-access-transfer-from-image function 652
- image-access-transfer-to-image function 653
- image-freed-p function 654
- :image-function initag 256, 290, 337, 569
- image-height accessor function 649
- :image-height initag 195, 552, 570
- image-list class 195
- :image-lists initag 256, 569
- image-loader function 654
- image-pinboard-object class 196
- image-pinboard-object-image accessor function 196
- :images initag 20, 552, 559

- image-set class 197
- :image-sets initarg 195
- image-translation function 655
- image-width accessor function 649
- :image-width initarg 195, 552, 570
- :init-function initarg 48
- :initial-focus initarg 206, 242
- :initial-focus-item initarg 35
- initialize-dithers function 656
- :in-place-completion-function initarg 524
- :in-place-filter initarg 524
- input focus 4
- :input-model initarg 341
- :insert-callback initarg 332
- inset-rectangle function 656
- inside-rectangle function 657
- installed-libraries function 200
- install-postscript-printer function 198
- :interaction initarg 15, 35
- interaction styles 17
- interactions
  - for choice 36
- interactive-pane class 201
- interactive-pane-execute-command generic function 203
- interactive-pane-stream accessor function 201
- interactive-pane-top-level-function accessor function 201
- interactive-stream 202
- interactive-stream-stream 202
- interactive-stream-top-level-function 202
- interface class 204
- :interface initarg 145
- interface-activate-callback accessor function 207
- interface-confirm-destroy-function accessor function 207
- interface-create-callback accessor function 207
- interface-customize-toolbar function 220
- interface-default-toolbar-states accessor function 207
- interface-destroy-callback accessor function 207
- interface-display generic function 221
- interface-display-title function 222
- interface-editor-pane generic function 223
- interface-extend-title generic function 223
- interface-geometry generic function 224
- interface-geometry-change-callback accessor function 207
- interface-help-callback accessor function 207
- interface-iconified-p function 225
- interface-iconify-callback accessor function 207, 215
- interface-keys-style generic function 225
- interface-match-p generic function 228
- interface-menu-bar-items accessor function 207
- interface-menu-groups generic function 229
- interface-message-area accessor function 207, 215
- interface-override-cursor accessor function 207
- interface-pointer-documentation-enabled accessor function 207
- interface-preserve-state generic function 230
- interface-preserving-state-p function 230
- interface-reuse-p generic function 231
- :interfaces class option 88
- :interfaces initarg 456
- interface-title accessor function 207
- interface-toolbar-items accessor function 207
- interface-toolbar-state function 232
- interface-toolbar-states accessor function 207
- interface-tooltips-enabled accessor function 207
- interface-visible-p function 234
- interface-window-styles accessor function 207
- internal scrolling 346
  - :internal-border initarg 498
  - :internal-max-height initarg 146, 370
  - :internal-max-width initarg 146, 370
  - :internal-min-height initarg 146, 370
  - :internal-min-width initarg 146, 370
- interpret-description generic function 235
- Interrupt playing a MIDI file 516

- invalidate-pane-constraints function 236
- invalidate-rectangle function 658
- invert-transform function 658
- invoke-command function 236
- invoke-untranslated-command function 237
- item class 237
- item-collection accessor function 238
- item-data accessor function 238
- itemp generic function 239
- item-pinboard-object class 239
- item-print-function accessor function 238
- :item-print-functions initarg 310
- :items initarg 52, 103, 158, 290, 295, 519, 577
- :items-count-function initarg 52, 158, 577
- item-selected accessor function 238
- :items-function initarg 290, 295
- :items-get-function initarg 52, 158, 577
- :items-map-function initarg 53, 158, 577
- item-text accessor function 238

## K

- :keep-selection-p initarg 35
- key press event handler 342
- :key-function initarg 519
- key-press events 342

## L

- labelled-arrow-pinboard-object class 240
- labelled-line-pinboard-object class 240
- labelled-line-text-foreground accessor function 241
- :label-style initarg 159
- :large-image-height initarg 257
- :large-image-width initarg 257
- layout class 241
- :layout initarg 204
- :layout-args initarg 20
- :layout-class initarg 20
- layout-description accessor function 242
- \*layout-divider-default-size\* 61, 454
- :layout-function initarg 178
- layout-ratios accessor function 60, 454
- :layouts class option 80

- layout-x-adjust accessor function 603
- :layout-x-adjust initarg 179
- layout-x-gap accessor function 189
- layout-x-ratios accessor function 189
- layout-y-adjust accessor function 603
- :layout-y-adjust initarg 179
- layout-y-gap accessor function 189
- layout-y-ratios accessor function 189
- :leaf-node-p-function initarg 570
- line-end-style graphics state parameter 668
- line-joint-style graphics state parameter 668
- line-pinboard-object class 243
- line-pinboard-object-coordinates function 244
- :line-size initarg 462
- :line-wrap-face initarg 132
- :line-wrap-marker initarg 132
- LispWorks as ActiveX control 87, 326
- list-all-font-names function 659
- listener-pane class 261
- listener-pane-insert-value function 262
- list-panel class 245
- list-panel-enabled generic function 252
- list-panel-filter-state generic function 253
- list-panel-items-and-filter function 254
- list-panel-right-click-selection-behavior accessor function 246
- list-panel-unfiltered-items generic function 255
- list-view class 256
- list-view-auto-reset-column-widths accessor function 258
- list-view-columns accessor function 258
- list-view-image-function accessor function 258
- list-view-state-image-function accessor function 258
- list-view-subitem-function accessor function 258
- list-view-subitem-print-functions accessor function 258
- list-view-view accessor function 258
- load-color-database function 732
- load-cursor function 263
- load-icon-image function 659
- load-image function 661
- load-sound function 266

locate-interface generic function 267  
lookup pane  
  interface 80  
lookup-pane 80  
lower-interface function 268

## M

Mac OS X Dock 45

macros

  define-color-models 726  
  define-command 77  
  define-interface 79  
  define-layout 86  
  define-menu 88  
  define-ole-control-component 87  
  display-errors 97  
  rectangle-bind 683  
  rectangle-bottom 684  
  rectangle-height 684  
  rectangle-left 685  
  rectangle-right 685  
  rectangle-top 686  
  rectangle-width 687  
  rect-bind 688  
  undefine-menu 582  
  union-rectangle 698  
  unless-empty-rect-bind 700  
  with-atomic-redisplay 587  
  with-busy-interface 587  
  with-dialog-results 588  
  with-dither 704  
  with-document-pages 591  
  with-external-metafile 592  
  with-geometry 594  
  with-graphics-mask 704  
  with-graphics-rotation 705  
  with-graphics-scale 706  
  with-graphics-state 707  
  with-graphics-transform 708  
  with-graphics-translation 709  
  with-internal-metafile 596  
  with-inverse-graphics 709  
  with-output-to-printer 597  
  without-relative-drawing 710  
  with-page 598  
  with-page-transform 598  
  with-pixmap-graphics-port 710  
  with-print-job 599  
  with-random-typeout 600  
  with-transformed-area 711  
  with-transformed-point 712  
  with-transformed-points 713

  with-transformed-rect 714  
make-container generic function 269  
make-dither function 664  
make-docking-layout-controller function 270  
make-font-description function 664  
make-foreign-owned-interface function 270  
make-general-image-set function 272  
make-graphics-state function 665  
make-gray function 733  
make-hsv function 734  
make-icon-resource-image-set function 273  
make-image function 670  
make-image-access generic function 671  
make-image-from-port function 672  
make-image-locator function 274  
make-menu-for-pane function 274  
make-pane-popup-menu generic function 275  
make-resource-image-set function 277  
make-rgb function 735  
make-scaled-general-image-set function 278  
make-scaled-image-set function 279  
make-sorting-description function 280  
make-sub-image function 673  
make-transform function 673  
manipulate-pinboard generic function 282  
map-collection-items generic function 285  
map-pane-children generic function 285  
map-pane-descendant-children generic function 288  
map-typeout function 289  
mask graphics state parameter 668  
mask-x graphics state parameter 668  
mask-y graphics state parameter 668  
:matches-title initarg 159  
:max-characters initarg 524  
%max-height% geometry slot 595  
\*maximum-moving-objects-to-track-edges\* variable 289  
%max-width% geometry slot 595  
MDI 68, 74, 109  
menu class 290  
:menu initarg 388  
:menu-bar class option 80  
:menu-bar-items initarg 204  
menu-component class 294

- :menu-function initarg 388
- menu-image-function accessor function 290
- menu-item class 297
- menu-items accessor function 290
- menu-object class 303
- menu-object-enabled accessor function 303
- menu-popup-callback accessor function 303
- :menus class option 80
- menu-title accessor function 545
- menu-title-function accessor function 545
- merge-font-descriptions function 674
- merge-menu-bars generic function 306
- :message initarg 546
- :message-area initarg 205
- :message-callback initarg 46
- :message-gap initarg 546
- message-pane class 307
- MIDI files
  - interrupting 516
- %min-height% geometry slot 594
- %min-width% geometry slot 594
- :mnemonic initarg 16, 21, 290, 297
- :mnemonic-escape initarg 16, 21, 290, 297
- :mnemonic-text initarg 16, 21
- :mnemonic-title initarg 21, 290, 297, 546
- modal dialogs 95, 381, 589
- modify-editor-pane-buffer function 308
- mono-screen class 309
- mouse clicks 342
- mouse coordinates 74
- mouse events 342
- mouse position 74
- move-line generic function 309
- multi-column-list-panel class 310
- multi-line-text-input-pane class 314
- Multiple Document Interface 68, 74, 109
- multiple-selection interaction style 17

## N

- :name initarg 30
- :navigation-callback initarg 525
- New in LispWorks 6.0
  - active-pane-copy 6
  - active-pane-copy-p 6
  - active-pane-cut 6
  - active-pane-cut-p 6
  - active-pane-deselect-all 6
  - active-pane-deselect-all-p 6
- active-pane-paste 6
- active-pane-paste-p 6
- active-pane-select-all 6
- active-pane-select-all-p 6
- active-pane-undo 6
- active-pane-undo-p 6
- current-popup 75
- drop-object-collection-index 123
- drop-object-collection-item 124
- element-widget-name 146
- interface-customize-toolbar 220
- interface-preserve-state 230
- interface-preserving-state-p 230
- interface-toolbar-state 232
- list-panel-filter-state 253
- list-panel-items-and-filter 254
- list-panel-unfiltered-items 255
- new-function 359
- non-focus-list-add-filter 316
- non-focus-list-interface 315
- non-focus-list-remove-filter 316
- non-focus-list-toggle-enable-filter 315
- non-focus-list-toggle-filter 316
- non-focus-maybe-capture-gesture 316
- non-focus-terminate 318
- non-focus-update 319
- pane-descendant-child-with-focus 356
- pane-interface-copy-object 359
- pane-interface-copy-p 359
- pane-interface-cut-object 359
- pane-interface-cut-p 359
- pane-interface-deselect-all 359
- pane-interface-deselect-all-p 359
- pane-interface-paste-object 359
- pane-interface-paste-p 359
- pane-interface-select-all 359
- pane-interface-select-all-p 359
- pane-interface-undo 359
- pane-interface-undo-p 359
- pane-supports-menus-with-images 363
- prompt-with-list-non-focus 319
- range-set-sizes 433
- scroll-if-not-visible-p 463
- set-editor-parenthesis-colors 474
- set-interactive-break-gestures 477
- stop-sound 516
- text-input-pane-in-place-complete 539

- :node-pane-function initarg 179
- :node-pinboard-class initarg 179
- non-focus-list-add-filter function 316
- non-focus-list-interface class 315
- non-focus-list-remove-filter function 316
- non-focus-list-toggle-enable-filter function 315
- non-focus-list-toggle-filter function 316
- non-focus-maybe-capture-gesture generic function 316
- non-focus-terminate generic function 318
- non-focus-update generic function 319
- no-selection interaction style 17
- :number initarg 456

## O

- %object% geometry slot 594
- offset-rectangle function 675
- :ok item in :buttons initarg 531
- ok-button image identifier 533
- OLE control 87, 326
- OLE embedding 87, 326
- ole-control-add-verbs function 324
- ole-control-close-object function 325
- ole-control-component class 326
- ole-control-component-pane accessor function 326
- ole-control-doc class 328
- ole-control-frame class 329
- ole-control-i-dispatch function 330
- ole-control-insert-object function 331
- ole-control-ole-object function 331
- ole-control-pane class 332
- ole-control-pane-frame function 335
- ole-control-pane-simple-sink class 335
- ole-control-user-component function 336
- operation graphics state parameter 666
- option-pane class 337
- option-pane-enabled accessor function 338
- option-pane-enabled-positions accessor function 338
- option-pane-image-function accessor function 338

- option-pane-popup-callback accessor function 338
- option-pane-separator-item accessor function 338
- option-pane-visible-items-count accessor function 338
- ordered-rectangle-union function 675
- ordinary scrolling 346
- :orientation initarg 103, 188, 432
- :orientation item in :buttons initarg 533
- output-pane class 340
- output-pane-create-callback accessor function 342
- output-pane-destroy-callback accessor function 342
- output-pane-display-callback accessor function 342
- output-pane-focus-callback accessor function 342
- output-pane-graphics-options accessor function 342
- output-pane-input-model accessor function 342
- output-pane-resize-callback accessor function 342
- output-pane-scroll-callback accessor function 342
- over-pinboard-object-p generic function 351
- :override-cursor initarg 205

## P

- page-setup-dialog function 351
- :page-size initarg 462
- pane-adjusted-offset generic function 352
- pane-adjusted-position generic function 353
- :pane-can-scroll initarg 341
- pane-close-display function 355
- pane-descendant-child-with-focus function 356
- :pane-function initarg 326
- pane-got-focus generic function 356
- pane-has-focus-p generic function 357
- pane-initial-focus generic function 357
- pane-interface-copy-object generic function 359
- pane-interface-copy-p generic function 359
- pane-interface-cut-object generic function 359

pane-interface-cut-p generic function 359  
pane-interface-deselect-all generic function 359  
pane-interface-deselect-all-p generic function 359  
pane-interface-paste-object generic function 359  
pane-interface-paste-p generic function 359  
pane-interface-select-all generic function 359  
pane-interface-select-all-p generic function 359  
pane-interface-undo generic function 359  
pane-interface-undo-p generic function 359  
pane-layout accessor function 21, 207  
:pane-menu initalg 498  
pane-popup-menu-items generic function 360  
:panes class option 80  
pane-string generic function 362  
pane-supports-menus-with-images function 363  
:paragraph-format initalg 444  
:parent initalg 145  
parse-layout-descriptor generic function 364  
password-pane class 364  
password-pane-overwrite-character accessor function 364  
pattern graphics state parameter 667  
:pinboard initalg 369  
pinboard-layout class 366  
pinboard-object class 369  
pinboard-object-activep accessor function 370  
pinboard-object-at-position generic function 372  
pinboard-object-graphics-arg generic function 373  
pinboard-object-graphics-args accessor function 370  
pinboard-object-overlap-p generic function 374  
pinboard-object-pinboard accessor function 370  
pinboard-pane-position generic function 375  
pinboard-pane-size generic function 376  
pixblt function 676  
pixmap-port class 677  
play-sound function 365  
:plist initalg 31  
:popup-callback initalg 303, 337, 523  
popup-confirmer function 377  
:popup-interface initalg 555  
popup-menu-button class 388  
popup-menu-button-menu accessor function 388  
popup-menu-button-menu-function accessor function 388  
port-height function 678  
port-string-height function 678  
port-string-width function 679  
port-width function 679  
:position item in :buttons initalg 533  
postmultiply-transforms function 680  
\*ppd-directory\* variable 397  
premultiply-transforms function 680  
:press-callback initalg 425  
print-capi-button generic function 388  
print-collection-item generic function 389  
print-dialog function 390  
print-editor-buffer function 392  
printer-configuration-dialog function 395  
printer-metrics structure type 396  
printer-metrics-device-height function 396  
printer-metrics-device-width function 396  
printer-metrics-dpi-x function 396  
printer-metrics-dpi-y function 396  
printer-metrics-height function 396  
printer-metrics-left-margin function 396  
printer-metrics-max-height function 396  
printer-metrics-max-width function 396  
printer-metrics-min-left-margin function 397  
printer-metrics-min-top-margin function 397  
printer-metrics-paper-height function 397  
printer-metrics-paper-width function 397  
printer-metrics-top-margin function 396  
printer-metrics-width function 396

printer-port-handle function 397  
 printer-port-supports-p function 398  
 \*printer-search-path\* variable 399  
 print-file function 392  
 :print-function initarg 52, 238, 519  
 print-rich-text-pane function 393  
 print-text function 394  
 process  
   CAPI 93  
   Cocoa Event Loop 93  
 process-pending-messages function 400  
 progress-bar class 401  
 prompt-for-color function 401  
 prompt-for-confirmation function 402  
 prompt-for-directory function 404  
 prompt-for-file function 406  
 prompt-for-files function 409  
 prompt-for-font function 410  
 prompt-for-form function 411  
 prompt-for-forms function 413  
 prompt-for-integer function 414  
 prompt-for-items-from-list function 415  
 prompt-for-number function 416  
 prompt-for-string function 417  
 prompt-for-symbol function 419  
 prompt-for-value function 421  
 prompt-with-list function 422  
 prompt-with-list-non-focus function 319  
 prompt-with-message function 424  
 :protected-callback initarg 444  
 push-button class 425  
 push-button-panel class 426

**Q**

quit-interface function 428

**R**

radio-button class 429  
 radio-button-panel class 430  
 raise-interface function 431  
 range-callback accessor function 433  
 range-end accessor function 433  
 range-orientation accessor function 433  
 range-pane class 432  
 range-set-sizes function 433  
 range-slug-end accessor function 433  
 range-slug-start accessor function 433  
 range-start accessor function 433  
 %ratio% geometry slot 594  
 :ratios initarg 60, 454  
 read-and-convert-external-image function 681  
 read-color-db function 736  
 read-external-image function 682  
 read-sound-file function 434  
 rectangle class 434  
 rectangle-bind macro 683  
 rectangle-bottom macro 684  
 rectangle-height macro 684  
 rectangle-left macro 685  
 rectangle-right macro 685  
 rectangle-top macro 686  
 rectangle-union function 686  
 rectangle-width macro 687  
 rect-bind macro 688  
 redisplay-collection-item generic function 435  
 redisplay-interface generic function 435  
 redisplay-menu-bar function 436  
 redraw-pinboard-layout function 437  
 redraw-pinboard-object function 437  
 register-image-load-function function 688  
 register-image-translation function 689  
 reinitialize-interface generic function 438  
 :remapped initarg 555  
 remove-capi-object-property function 438  
 remove-items generic function 439  
 replace-dialog function 440  
 replace-items generic function 441  
 report-active-component-failure generic function 442  
 reset-image-translation-table function 690  
 resizable  
   dialogs 213  
   elements 149  
   windows 208  
 :resize-callback initarg 341  
 resizing 149, 208, 213  
 :retain-expanded-nodes initarg 570  
 :retract-callback initarg 17, 27  
 Return key 379  
 reuse-interfaces-p function 443

- rich-text-pane class 444
- rich-text-pane-change-callback accessor function 444
- rich-text-pane-character-format function 446
- rich-text-pane-limit accessor function 444
- rich-text-pane-operation function 447
- rich-text-pane-paragraph-format function 451
- rich-text-pane-text accessor function 444
- rich-text-version function 452
- right-angle-line-pinboard-object class 452
- :right-click-extended-match initarg 570
- :right-click-selection-behavior initarg 245
- :roots initarg 178, 569
- row-layout class 453
- row-layout-divider 454
- :rows initarg 189

## S

- :save-name initarg 332
- scale-thickness graphics state parameter 667
- screen
  - usable region of 458
- screen class 455
- screen-active-interface function 457
- screen-active-p function 457
- screen-depth accessor function 456
- screen-height accessor function 456
- screen-height-in-millimeters accessor function 456
- screen-interfaces accessor function 108, 456
- screen-internal-geometry function 458
- screen-logical-resolution function 458
- screen-number accessor function 456
- screens function 459
- screen-width accessor function 456
- screen-width-in-millimeters accessor function 456
- scroll generic function 460
- scroll-bar class 462
- scroll-bar-line-size accessor function 462
- scroll-bar-page-size accessor function 462

- :scroll-callback initarg 341
- %scroll-height% geometry slot 595
- %scroll-horizontal-page-size% geometry slot 595
- %scroll-horizontal-slug-size% geometry slot 595
- %scroll-horizontal-step-size% geometry slot 595
- scroll-if-not-visible-p generic function 463
- :scroll-if-not-visible-p initarg 498
- scrolling
  - built-in 174
  - internal 346
  - ordinary 346
- %scroll-start-x% geometry slot 595
- %scroll-start-y% geometry slot 595
- %scroll-vertical-page-size% geometry slot 595
- %scroll-vertical-slug-size% geometry slot 595
- %scroll-vertical-step-size% geometry slot 595
- %scroll-width% geometry slot 595
- %scroll-x% geometry slot 595
- %scroll-y% geometry slot 595
- search-for-item generic function 465
- :selected initarg 15, 238
- :selected-disabled-image initarg 16
- :selected-disabled-images initarg 21
- :selected-function initarg 297
- :selected-image initarg 15, 554
- :selected-images initarg 21
- :selected-item initarg 35, 577
- :selected-item-function initarg 295
- :selected-items initarg 35
- :selected-items-function initarg 295
- selection function 465
- :selection initarg 35
- :selection-callback initarg 17, 27, 519
- selection-empty function 466
- :selection-function initarg 295
- separation function 691
- :separator-item initarg 337
- set-application-interface function 467
- set-button-panel-enabled-items
  - generic function 468
- set-clipboard function 468
- set-confirm-quit-flag function 470
- set-default-editor-pane-blink-rate function 470
- set-default-image-load-function

- function 691
- set-default-interface-prefix-suffix function 471
- set-drop-object-supported-formats function 473
- set-editor-parenthesis-colors function 474
- set-geometric-hint function 475
- set-graphics-port-coordinates function 692
- set-graphics-state function 693
- set-hint-table function 475
- set-horizontal-scroll-parameters generic function 476
- set-interactive-break-gestures function 477
- set-object-automatic-resize function 478
- set-pane-focus generic function 482
- set-printer-metrics function 488
- set-printer-options function 489
- set-rich-text-pane-character-format function 482
- set-rich-text-pane-paragraph-format function 485
- set-scroll-position generic function 461
- set-scroll-range generic function 476, 493
- set-selection function 487
- set-text-input-pane-selection function 490
- set-top-level-interface-geometry generic function 491
- :setup-callback-argument initag 303
- set-vertical-scroll-parameters generic function 492
- shell-pane class 493
- shell-pane-command accessor function 494
- show-interface function 495
- show-pane function 495
- :show-value-p initag 510, 512
- simple-layout class 496
- simple-network-pane class 496
- simple-pane class 497
- simple-pane-background accessor function 499
- simple-pane-cursor accessor function 499
- simple-pane-drag-callback accessor function 499
- simple-pane-drop-callback accessor function 499
- simple-pane-enabled accessor function 159, 499, 561
- simple-pane-font accessor function 499
- simple-pane-foreground accessor function 499
- simple-pane-handle function 505
- simple-pane-horizontal-scroll accessor function 499
- simple-pane-scroll-callback accessor function 499
- simple-pane-vertical-scroll accessor function 499
- simple-pane-visible-border accessor function 499
- simple-pane-visible-height generic function 506
- simple-pane-visible-size generic function 506
- simple-pane-visible-width generic function 507
- simple-pinboard-layout class 508
- simple-print-port function 509
- single-selection interaction style 17
- :sinks initag 332
- slider class 510
- slider-show-value-p accessor function 510
- slider-start-point accessor function 510
- :slug-end initag 432
- :slug-start initag 432
- :small-image-height initag 257
- :small-image-width initag 257
- sorted-object class 512
- sorted-object-sort-by generic function 512
- sort-object-items-by function 511
- :source-interfaces class option 88
- special slots
  - container 109
  - windows-menu 109
- :start initag 432, 542
- start-gc-monitor function 513
- :start-point initag 510
- :start-x initag 244
- :start-y initag 244
- :state-image-function initag 256, 569
- :state-image-height initag 257, 570
- :state-image-width initag 257, 570
- static-layout class 514

- stipple graphics state parameter 666
- stop-gc-monitor function 515
- stop-sound function 516
- :stream initarg 58
- streams 58
- :stretch-text-p initarg 552
- structure types
  - printer-metrics 396
- :subitem-function initarg 256
- :subitem-print-functions initarg 256
- switchable-layout class 516
- switchable-layout-combine-child-constraints accessor function 517
- switchable-layout-switchable-children generic function 518
- switchable-layout-visible-child accessor function 517

## T

- tab-layout class 518
- tab-layout-combine-child-constraints accessor function 519
- tab-layout-panes function 521
- tab-layout-visible-child function 522
- tab-layout-visible-child-function accessor function 519
- tabstops 4
- :test-function initarg 52
- :text initarg 99, 131, 159, 238, 444, 524, 542, 544
- :text-change-callback initarg 525
- :text-foreground initarg 241
- text-input-choice class 522
- text-input-pane class 523
- text-input-pane-before-change-callback accessor function 534
- text-input-pane-buttons-enabled accessor function 525
- text-input-pane-callback accessor function 525
- text-input-pane-caret-position accessor function 525
- text-input-pane-change-callback accessor function 525
- text-input-pane-complete-text function 536
- text-input-pane-completion-function accessor function 525
- text-input-pane-confirm-change-function accessor function 525
- text-input-pane-copy function 537

- text-input-pane-cut function 538
- text-input-pane-delete function 538
- text-input-pane-editing-callback accessor function 525
- text-input-pane-enabled accessor function 525
- text-input-pane-in-place-complete function 539
- text-input-pane-max-characters accessor function 525
- text-input-pane-navigation-callback accessor function 525
- text-input-pane-paste function 539
- text-input-pane-selected-text function 540
- text-input-pane-selection function 540
- text-input-pane-selection-p function 541
- text-input-pane-text accessor function 525
- text-input-range class 542
- text-input-range-callback accessor function 542
- text-input-range-callback-type accessor function 542
- text-input-range-end accessor function 542
- text-input-range-start accessor function 542
- text-input-range-value accessor function 542
- text-input-range-wraps-p accessor function 542
- :text-limit initarg 444
- thickness graphics state parameter 667
- title bar
  - removal 212
- :title initarg 204, 545, 546
- :title-adjust initarg 168, 546
- :title-args initarg 546
- titled-menu-object class 544
- titled-object class 546
- titled-object-message accessor function 547
- titled-object-message-font accessor function 215, 547
- titled-object-title accessor function 547
- titled-object-title-font accessor function 547
- titled-pane 548
- titled-pane-message 548

- titled-pane-title 548
- titled-pinboard-object class 549
- :title-font initarg 546
- :title-function initarg 545
- :title-gap initarg 168, 546
- title-pane class 543
- title-pane-text accessor function 544
- :title-position initarg 546
- :to initarg 176
- toolbar class 552
- toolbar-button class 554
- toolbar-button-dropdown-menu accessor function 555
- toolbar-button-dropdown-menu-function accessor function 555
- toolbar-button-dropdown-menu-kind accessor function 555
- toolbar-button-image accessor function 555
- toolbar-button-popup-interface accessor function 555
- toolbar-button-selected-image accessor function 555
- toolbar-component class 559
- toolbar-flat-p accessor function 552
- :toolbar-items initarg 206
- toolbar-object class 561
- toolbar-object-enabled-function accessor function 561
- :toolbar-states initarg 206
- :toolbar-title initarg 498
- :tooltip initarg 554
- :tooltips initarg 552, 560
- :top-level-function initarg 201
- :top-level-hook initarg 206
- top-level-interface generic function 561
- top-level-interface-display-state generic function 562
- top-level-interface-external-border accessor function 207
- top-level-interface-geometry generic function 563
- top-level-interface-geometry-key generic function 564
- top-level-interface-p generic function 566
- top-level-interface-save-geometry-p generic function 566
- top-level-interface-transparency accessor function 207
- tracking-pinboard-layout class 567
- transform graphics state parameter 665
- transform type 693
- transform-area function 694
- transform-distance function 694
- transform-distances function 695
- transform-is-rotated function 695
- transform-point function 696
- transform-points function 696
- transform-rect function 697
- :transparency initarg 206
- tree-view class 569
- tree-view-action-callback-expand-p accessor function 572
- tree-view-checkbox-change-callback accessor function 572
- tree-view-checkbox-child-function accessor function 572
- tree-view-checkbox-initial-status accessor function 572
- tree-view-checkbox-next-map accessor function 572
- tree-view-checkbox-parent-function accessor function 572
- tree-view-checkbox-status accessor function 572
- tree-view-children-function accessor function 572
- tree-view-ensure-visible function 578
- tree-view-expanded-p generic function 578
- tree-view-expandp-function accessor function 572
- tree-view-has-root-line accessor function 572
- tree-view-image-function accessor function 572
- tree-view-item-checkbox-status function 579
- tree-view-item-children-checkbox-status function 580
- tree-view-leaf-node-p-function accessor function 572
- tree-view-retain-expanded-nodes accessor function 572
- tree-view-right-click-extended-match accessor function 572
- tree-view-roots accessor function 572
- tree-view-state-image-function accessor function 572
- tree-view-update-an-item generic function 580
- tree-view-update-item generic function

581  
:type initarg 453  
types  
  transform 693

## U

unconvert-color function 737  
undefine-font-alias function 698  
undefine-menu macro 582  
unhighlight-pinboard-object generic  
  function 582  
:uniform-size-p initarg 60, 454  
uninstall-postscript-printer func-  
  tion 582  
union-rectangle macro 698  
\*unit-transform\* variable 699  
unit-transform-p function 699  
unless-empty-rect-bind macro 700  
unmap-typeout function 583  
untransform-distance function 700  
untransform-distances function 701  
untransform-point function 701  
untransform-points function 702  
update-all-interface-titles function  
  584  
update-interface-title generic func-  
  tion 584  
update-pinboard-object function 585  
\*update-screen-interfaces-hooks\*  
  variable 586  
update-screen-interface-titles func-  
  tion 585  
update-toolbar function 586  
:use-images initarg 570  
:use-large-images initarg 257  
:user-component initarg 332  
:use-small-images initarg 257  
:use-state-images initarg 257, 570

## V

validate-rectangle function 703  
variables  
  \*color-database\* 721  
  \*default-editor-pane-line-wrap-  
    marker\* 76  
  \*default-image-translation-table\*  
    616, 656  
  \*echo-area-cursor-inactive-style\*  
    128  
  \*editor-cursor-active-style\* 130  
  \*editor-cursor-color\* 129

  \*editor-cursor-drag-style\* 130  
  \*editor-cursor-inactive-style\* 130  
  \*maximum-moving-objects-to-track-  
    edges\* 289  
  \*ppd-directory\* 397  
  \*printer-search-path\* 399  
  \*unit-transform\* 699  
  \*update-screen-interfaces-hooks\*  
    586  
:vertical-adjustment initarg 168  
:vertical-gap initarg 168  
:vertical-scroll initarg 463, 498  
:view initarg 256  
:view-class initarg 48  
:visible-border initarg 498  
:visible-child initarg 516  
:visible-child-function initarg 519  
:visible-items-count initarg 337, 523  
:visible-max-height initarg 146, 370  
:visible-max-width initarg 146, 370  
:visible-min-height initarg 146, 370  
:visible-min-width initarg 146, 370

## W

:widget-name initarg 145  
%width% geometry slot 594  
:width initarg 456  
windoid 213  
Window handle 72, 505  
window title  
  removal 212  
window-modal dialogs 95, 381, 589  
windows-menu 110  
windows-menu special slot 109  
:window-styles initarg 206  
with-atomic-redisplay macro 587  
with-busy-interface macro 587  
with-dialog-results macro 588  
with-dither macro 704  
with-document-pages macro 591  
with-external-metafile macro 592  
with-geometry macro 594  
with-graphics-mask macro 704  
with-graphics-rotation macro 705  
with-graphics-scale macro 706  
with-graphics-state macro 707  
with-graphics-transform macro 708  
with-graphics-translation macro 709  
with-internal-metafile macro 596  
with-inverse-graphics macro 709  
with-output-to-printer macro 597  
without-relative-drawing macro 710

- with-page macro 598
- with-page-transform macro 598
- with-pixmap-graphics-port macro 710
- with-print-job macro 599
- with-random-typeout macro 600
- with-transformed-area macro 711
- with-transformed-point macro 712
- with-transformed-points macro 713
- with-transformed-rect macro 714
- :wraps-p initarg 542
- :wrap-style initarg 132
- wrap-text function 600
- wrap-text-for-pane function 601
- write-external-image function 714

## X

- %x% geometry slot 594
- :x initarg 145, 369
- X window ID 72, 505
- X Window System
  - display 68
  - fallback resources 68
- :x-adjust initarg 603
- :x-gap initarg 189, 496
- :x-ratios initarg 189
- :x-uniform-size-p initarg 189
- x-y-adjustable-layout class 602

## Y

- %y% geometry slot 594
- :y initarg 145, 369
- :y-adjust initarg 603
- :y-gap initarg 189
- :y-ratios initarg 189
- :y-uniform-size-p initarg 189

## Z

- Z-order
  - of interfaces 52
  - of pinboard-objects 243, 367