Contents

Preface  xxi

1  CAPI Reference Entries  1
   abort-callback 1
   abort-dialog 2
   abort-exit-confirm 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
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-confirm 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-colormap 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
make-sub-image 673
make-transform 673
merge-font-descriptions 674
offset-rectangle 675
ordered-rectangle-union 675
pixblt 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 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 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 if you find any mistakes in the LispWorks documentation, or if you have any suggestions for improvements.
The following chapter documents symbols exported from the `capi` package.

**abort-callback**

Function

Summary Aborts out of the context of the current callback.

Package `capi`

Signature `abort-callback &optional always-abort`

Arguments `always-abort` A generalized boolean.

Description The function `abort-callback` aborts out of the context of the current callback, returning `nil` when it is relevant (for example in an interface `confirm-destroy-callback`).

If called outside the context of a callback, if `always-abort` is `t` then `abort-callback` calls `(abort)`, otherwise it just returns.

The default value of `always-abort` is `t`. 
See also  callbacks
interface

abort-dialog

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.
(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-claimer
interface

Abort-exit-claimer

Function

Summary
  Aborts the exiting of a dialog.

Package
capi

Signature
  abort-exit-claimer

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

  If abort-exit-claimer is called outside the exiting of a claimer, 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-confirmers
 (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-confirmers))
    value))

See also popup-confirmers

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.
activate-pane

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
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.

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`.

7
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**

<table>
<thead>
<tr>
<th>Class</th>
</tr>
</thead>
</table>

**Summary**

A pinboard-object that draws itself as an arrow.

**Package**

capi

**Superclasses**

line-pinboard-object

**Subclasses**

double-headed-arrow-pinboard-object

labelled-arrow-pinboard-object

**Initargs**

- **:head**
  A keyword specifying the position of the arrowhead on the line.

- **:head-direction**
  A keyword specifying the direction of the arrowhead.

- **:head-length**
  The length of the arrowhead.

- **:head-breadth**
  The breadth of the arrowhead, or nil.

- **:head-graphics-args**
  A graphics args plist.

**Description**

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

head must be :end, :middle or :start. The default is :end.
head-direction must be :forwards, :backwards or :both. The default is :forwards.

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

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

head-graphics-args is a plist of graphics state parameters and values used when drawing the arrow head. For information about the graphics state, see make-graphics-state.
Example

(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 capi

Signature attach-interface-for-callback element interface
The function `attach-interface-for-callback` changes the interface that is passed when a callback is made. Callbacks for `element` get passed `interface` instead of `element`'s parent interface.

See also
- callbacks
- `element`
- `element-interface-for-callback`
- `interface`

### `attach-simple-sink`

**Function**

**Summary**
Attaches a sink to the active component in an `ole-control-pane`.

**Package**
capi

**Signature**

```lisp
attach-simple-sink invoke-callback pane interface-name &key sink-class => sink
```

**Arguments**

- `invoke-callback` A function designator.
- `pane` An `ole-control-pane`.
- `interface-name` A refguid or the symbol `:default`.
- `sink-class` A symbol naming a class.

**Values**

- `sink` The sink object.

**Description**

The function `attach-simple-sink` make a sink object and attaches it to the active component in `pane`.

When an event callback is triggered for the source interface named by `interface-name`, the sink object will call the `invoke-callback` with four arguments: the `pane` (see `sink-class` below), the source method name as a string, the source method type (either `:method`, `:get` or `:put`) and a vector of the remaining callback arguments.
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 capi

Signature beep-pane &optional pane

Description The function beep-pane sounds a beep on the screen associated with pane or on the current screen if pane is nil.

Example (capi:beep-pane)

See also simple-pane screen
**button**

**Summary**
A button 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**
capi

**Superclasses**
simple-pane
item

**Subclasses**
push-button
radio-button
check-button

**Initargs**

- **:interaction** The interaction style for the button.
- **:selected** For radio button and check button styles, if selected is set to t, the button is initially selected.
- **:callback** Specifies the callback to use when the button is selected.
- **:image** An image for the button (or nil).
- **:selected-image** The image used when the button is selected.
- **:enabled** If nil the button cannot be selected.
- **:cancel-p** If true the button is the "Cancel" button, that is, the button selected by the Escape key.
- **:default-p** If true the button is the default button, that is, the button selected by the Return key.

The following two initargs controlling alternate images apply only on Motif:

- **:disabled-image** The image for the button when disabled (or nil).
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 runtime) 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.
A list.

A list.

A list.

Specifies the default button.

Specifies the cancel button.

The following initargs controlling mnemonics apply only on Microsoft Windows:

A list specifying mnemonics for the buttons.

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

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

A string specifying the title and a mnemonic.

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:
For example,

```lisp
(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`.

<table>
<thead>
<tr>
<th>Button panel class</th>
<th>Button class</th>
</tr>
</thead>
<tbody>
<tr>
<td>push-button-panel</td>
<td>push-button</td>
</tr>
<tr>
<td>radio-button-panel</td>
<td>radio-button</td>
</tr>
<tr>
<td>check-button-panel</td>
<td>check-button</td>
</tr>
</tbody>
</table>

**Table 1.1 Button and panel classes**
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

```lisp
(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
capi

Signature
calculate-constraints pane

Arguments
pane A CAPI pane or layout.

Description
The generic function calculate-constraints calculates the minimum and maximum size for pane according to the sizes of its children, and sets these values into pane's geometry cache.

The CAPI calls calculate-constraints for each pane and layout that it displays.

When creating your own layout, you should define a method for calculate-constraints that sets the values of the following geometry slots based on the constraints of its children.

%min-width% The minimum width of pane.
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.
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:

See also
get-constraints
with-geometry
interpret-description
function   Call the function.
list   Apply the head of the list to the tail.

:redisplay-interface
Call \texttt{redisplay-interface} on the top-level interface.

:redisplay-menu-bar
Call \texttt{redisplay-menu-bar} on the top-level interface.

The slot value \texttt{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
\hspace{1em} (collection data)
::data
\hspace{1em} (item-data)
::data-element
\hspace{1em} (item-data element)
::data-interface
\hspace{1em} (item-data interface)
::element
\hspace{1em} (element)
::element-data
\hspace{1em} (element item-data)
::element-item
\hspace{1em} (element item)
::interface-data
\hspace{1em} (interface item-data)
::item
\hspace{1em} (item)
::item-element
\hspace{1em} (item element)
::item-interface
\hspace{1em} (item interface)
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

```lisp
(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.
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
```lisp
(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`.

```lisp
(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

```lisp
(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

button-panel
extended-selection-tree-view
graph-pane
list-panel
menu-component
option-pane
tree-view

Initargs

:interaction  The interaction style of the choice.

:selection    The indexes of the choice's selected items.

:selected-item

The selected item for a single selection choice.

:selected-items

A list of the selected items.

:keep-selection-p

If t, retains any selection when the items change.

:initial-focus-item

If supplied, this should be an item in the choice.

Accessors

choice-selection

Readers

choice-interaction
choice-initial-focus-item

Description

The class choice inherits most of its behavior from collection, and then provides the selection facilities itself. The classes list-panel, button-panel, option-pane, menu-component and graph-pane inherit from it, and so it plays a key role in CAPI applications.

A choice can have one of four different interaction styles, and these control how it behaves when an item is selected by the user. interaction can be one of:
:no-selection  The choice behaves just as a collection.
:single-selection
   The choice can have only one selected item.
:multiple-selection
   The choice can have multiple selected items.
:extended-selection
   An alternative to multiple-selection.

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

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

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

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

On Mac OS X, the selection gesture is mouse (left button) click. De-selection and discontinuous selections are made by Command+Click, and a continuous selection is made by Shift+Click, regardless of whether if interaction is :multiple-selection or :extended-selection.
The choice’s selection stores the indices of the currently selected item, and is a single number for single selection choices and a list for all other interactions. The functions choice-selected-item and choice-selected-items treat the selection in terms of the items themselves as opposed to their indices.

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

However, if the choice was created with :keep-selection-p, 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
Generic Function

choice-selected-item

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`.

```lisp
(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.

```lisp
(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.
The function **choice-selected-item-p** is the predicate for whether an item `item` of the choice `choice` is selected.

Note that the items are compared by `choice`'s `test-function` - see `collection` for details.

**Example**

```lisp
(setq list
  (capi:contain
    (make-instance 'capi:list-panel
      :items '(a b c d)
      :selection 2
      :visible-min-height
      '(:character 4))))

(capi:choice-selected-item-p list 'c)
=>
t

Now click on another item.

(capi:choice-selected-item-p list 'c)
=>
nil
```

**See also**

choice
collection

---

**choice-selected-items**

Generic Function

The function **choice-selected-items** returns the currently selected items in a choice as a list of the items.

**Summary**

The function **choice-selected-items** returns the currently selected items in a choice as a list of the items.

**Package**
capi

**Signature**

choice-selected-items choice

**Signature**

(setf choice-selected-items) items choice
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.

```lisp
(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.

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

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

```lisp
(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 | capi  
---|---  
Signature | choice-update-item choice item  
Description | The function choice-update-item updates the display of the item item in the choice choice. It should be called if the display of item (that is, the string returned by the print-function) 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  
capi

Signature  
\texttt{clipboard self \&optional format =\rightarrow result}

Arguments  
\texttt{self} \hspace{1cm} \text{A displayed CAPI pane or interface.}
\texttt{format} \hspace{1cm} \text{A keyword.}

Values  
\texttt{result} \hspace{1cm} \text{A string, an \texttt{image}, a Lisp object, or \texttt{nil}.}

Description  
The function \texttt{clipboard} returns the contents of the system clipboard as a string, or \texttt{nil} if the clipboard is empty.

\texttt{format} controls what kind of object is read. The following values of \texttt{format} are recognized:

\begin{itemize}
  \item \texttt{:string} \hspace{1cm} \text{The object is a string. This the default value.}
  \item \texttt{:image} \hspace{1cm} \text{The object is of type \texttt{image}, converted from whatever format the platform supports.}
  \item \texttt{:value} \hspace{1cm} \text{The object is the Lisp value.}
  \item \texttt{:metafile} \hspace{1cm} \text{The object is a metafile.}
\end{itemize}

When \texttt{format} is \texttt{:image}, the image returned by \texttt{clipboard} is associated with \\texttt{self}, so you can free it explicitly with \texttt{free-image} or it will be freed automatically when the pane is destroyed.

When \texttt{format} is \texttt{:metafile} the object is a metafile which should be freed using \texttt{free-metafile} when no longer needed. See also \texttt{draw-metafile} and \texttt{draw-metafile-to-image}. \texttt{format :metafile} is not supported on GTK+ or X11/Motif.

The Microsoft Windows clipboard is usually set by the user with the Ctrl+C and Ctrl+X gestures. Note that the Lisp-Works editor uses these gestures when in Windows emulation mode.
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**
capi

**Signature**
`clipboard-empty self &optional format => result`

**Arguments**
- `self` A displayed CAPI pane or interface.
- `format` A keyword.

**Values**
- `result` `t` or `nil`.

**Description**
The function `clipboard-empty` returns `nil` if there is an object of the kind indicated by `format` on the clipboard, or `t` otherwise.
format controls what kind of object is checked. The allowed values of format are as described for clipboard.

See also clipboard, image

c**lone**

*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

:message-callback
   A function or \texttt{nil}.

:application-menu
    \texttt{nil}, a \texttt{menu}, or the name of a slot containing
    a \texttt{menu} in the application interface.

:dock-menu
   \texttt{nil}, a \texttt{menu}, or a function designator.

Accessors

\texttt{application-interface-message-callback}
\texttt{application-interface-application-menu}
\texttt{application-interface-dock-menu}

Description

The class \texttt{cocoa-default-application-interface} supports
application messages and the application menu for a Cocoa
application.

When non-nil, \texttt{message-callback} should be a function with sig-
nature

\texttt{interface message \&rest args}

\texttt{message-callback} will be called for various application
messages. The \texttt{interface} argument will be the application
interface and the \texttt{message} argument will be a keyword. The
only currently defined message is \texttt{:open-file}. In this case
\texttt{args} 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.

\texttt{application-menu} controls the application’s main menu. If this
is \texttt{nil}, then a minimal application menu will be made using
the title of the application interface, otherwise it should be a
\texttt{menu} containing the usual items or the name of a slot
containing such a menu in the application interface.

\texttt{dock-menu} provides a menu for use by the Mac OS X Dock
icon. If the value is \texttt{nil} (the default), then the standard menu
is used. If \texttt{dock-menu} is a function designator, it is called with
the application interface as its argument when the menu is
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:

- **:normal** Show the application and activate it
- **:restore** Show the application again without activating it
- **:hidden** Hide
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
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 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.


### 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

\textit{cocoa-view-pane-view} \hspace{1cm} \textbf{Function}

\textbf{Summary} \hspace{1cm} Returns the Cocoa view of a \texttt{cocoa-view-pane}.

\textbf{Package} \hspace{1cm} capi

\textbf{Signature} \hspace{1cm} \texttt{capi\textbackslash{}cocoa-view-pane-view pane} $\Rightarrow$ \texttt{view}

\textbf{Arguments} \hspace{1cm} \texttt{pane} \hspace{1cm} A \texttt{cocoa-view-pane}.

\textbf{Values} \hspace{1cm} \texttt{view} \hspace{1cm} A foreign pointer to a Cocoa view or \texttt{nil}.

\textbf{Description} \hspace{1cm} The function \texttt{cocoa-view-pane-view} returns the Cocoa view for the \texttt{cocoa-view-pane pane} as a foreign pointer. This view is only accessible when the pane is visible and \texttt{nil} is returned in other cases.

\textit{Note}: \texttt{cocoa-view-pane-view} is implemented only in Lisp-Works for Macintosh with the Cocoa IDE. See the \textit{LispWorks Objective-C and Cocoa Interface User Guide and Reference Manual} for details on using Cocoa.

\textbf{Example} \hspace{1cm} See the example in \texttt{examples/objc/movie-view.lisp}.

See also \hspace{1cm} \texttt{cocoa-view-pane}
**collect-interfaces**

*Generic Function*

**Summary**
Finds all interfaces of a given class.

**Package**
capi

**Signature**

```
collect-interfaces proto &key screen current-process-first sort-by => interfaces
```

**Arguments**

- `proto` A class, class name, or an interface.
- `screen` nil, the symbol :any, a screen, or a keyword naming a library.
- `current-process-first` A boolean.
- `sort-by` :visible or :create.

**Values**

- `interfaces` A list.

**Description**
The generic function `collect-interfaces` returns a list of CAPI interfaces which are instances of the class indicated by `proto`, or subclasses thereof.

If `screen` is `nil`, the interfaces on the default screen are returned. This is the default. If `screen` is `:any`, `interfaces` includes those on any screen. If `screen` is a `screen` object, the interfaces on that screen are returned. `screen` can also be a library name, currently the accepted values are :win32, :motif and :cocoa.

If interfaces on multiple screens are returned, then those on each screen are grouped together in `interfaces`.

Amongst those for each screen, the interfaces are grouped as follows. If `current-process-first` is true, then the interfaces in the current process appear together at the beginning of the group. If `sort-by` is :create then these interfaces are sorted by
creation time, otherwise sort-by is visible and they are 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.
collections

: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 interpretted 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 collection &key set => index

Arguments
collection A collection.
set A boolean.
The generic function `collection-find-next-string` must be called after one of `collection-search`, `collection-find-string` or `find-string-in-collection` was called on `collection`. It searches for the next item in `collection` with printed representation matching the last string searched for and returns its index, or `nil` if no match is found.

If `set` is true, then if an item matching the string is found, the selection is set to this item. `set` defaults to `t`.

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

---

**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**  
capi

**Signature**
`collection-find-string collection &key set string => index`

**Arguments**
- `collection`  
  A collection.
- `set`  
  A boolean.
- `string`  
  A string, or `nil`.

**Values**
`index`  
A non-negative integer or `nil`.

**Description**
The generic function `collection-find-string` calls `find-string-in-collection` with `collection` and `set`.  

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

collector-pane

Class

Summary        A collector-pane is an editor-pane 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       capi
Superclasses   editor-pane
Initargs       :buffer-name   The name of a buffer onto an editor stream.
                :stream       The editor stream to be collected.
Readers        collector-pane-stream

Description    A new collector-pane 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 buffer-name which does not match any existing buffer, or do not pass buffer-name 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).

\[
\begin{align*}
\text{(setq collector (capi:contain } & \text{(make-instance 'capi:collector-pane)))} \\
\text{(setq *test-stream* } & \text{(capi:collector-pane-stream collector))} \\
\text{(capi:contain } & \text{(make-instance 'capi:collector-pane :stream *test-stream*)))} \\
\text{(format *test-stream* "Hello World~")}
\end{align*}
\]

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

\[
\begin{align*}
\text{(capi:contain (make-instance 'capi:collector-pane :buffer-name "Background Output"))}
\end{align*}
\]

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  
\begin{itemize}
  \item \texttt{element-screen}
  \item \texttt{mono-screen}
\end{itemize}

\textbf{column-layout}  
\textit{Class}

\textbf{Summary}
\texttt{The column-layout lays its children out in a column.}

\textbf{Package}
\texttt{capi}

\textbf{Superclasses}
\texttt{grid-layout}

\textbf{Initargs}
\begin{itemize}
  \item \texttt{:ratios} \hspace{1em} The size ratios between the layout’s children.
  \item \texttt{:adjust} \hspace{1em} The horizontal adjustment for each child.
  \item \texttt{:gap} \hspace{1em} The gap between each child.
  \item \texttt{:uniform-size-p} \hspace{1em} If \texttt{t}, each child in the column has the same height.
\end{itemize}

\textbf{Accessors}
\texttt{layout-ratios}

\textbf{Description}
\texttt{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.}

\texttt{description} may also contain the keywords :\texttt{divider} and :\texttt{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**

```lisp
(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 `component-name` of an `ole-control-pane`.

**Package**
capi

**Signature**

```
component-name pane => name
(setf component-name) name pane => name
```

**Description**
The function `component-name` accesses the `component-name` of an `ole-control-pane`.

When the `ole-control-pane` is created, it automatically opens the component and inserts it.

If `(setf component-name)` is called on a pane that is already created, any existing component is closed, and the new component is opened and inserted. `(setf component-name)` also sets the pane’s `user-component` to `nil`.

**Note:** `component-name` 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

---

**confirm-quit**

*Function*

**Summary**
Quits the Lisp session, potentially after user confirmation.

**Package**
capi

**Signature**

```
confirm-quit application-name
```

**Arguments**

- `application-name` A string.
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 `confirm-yes-or-no` pops up a dialog button containing a message and a **Yes** and **No** button.

**Package**
capi

**Signature**
`confirm-yes-or-no format-string &rest format-args`

**Description**
This pops up a dialog box containing a message and the buttons **Yes** and **No**, returns `t` when the **Yes** button is clicked, and `nil` when the **No** button is clicked. The message is obtained by applying the `format-string` and the `format-args` to the Common Lisp function `format`.

This function is actually a convenient version of `prompt-for-confirmation`, but has the disadvantage that you cannot specify any customization arguments. For more flexibility, use `prompt-for-confirmation` itself.

**Example**
```lisp
(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))
```

**See also**
`prompt-for-confirmation`
`display-dialog`
`popup-confirm`r

**confirmer-pane**

**Function**

**Summary**
Returns the pane associated with a confirmer interface.

**Package**
capi
### confirm-pane

**Signature**

```
confirm-pane interface => pane
```

**Arguments**

*interface*  
A confirmer interface displayed by *popup-confirmer*.

**Values**

*pane*  
The *pane* argument passed to *popup-confirmer*.

**Description**

The function *confirm-pane* returns the pane associated with a confirmer interface that has been displayed by *popup-confirmer*.

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 *buttons* argument.

**See also**

*popup-confirmer*

---

### contain

**Summary**

Displays a window containing an element.

**Package**

capi

**Signature**

```
contain element &rest interface-args &key screen process title &allow-other-keys => element
```

**Description**

The function *contain* creates and displays a container for the CAPI element *element*. *contain* returns *element* as its result.

*contain* is provided as a convenient way of testing CAPI functionality and is useful mainly during interactive development. Many of the CAPI examples use it.

The container is created using *make-container*, which can make containers for any of the following classes:
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

```lisp
(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
**convert-relative-position**

**Signature**

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

**Arguments**

- `from` A pane, interface or screen.
- `to` A pane, interface or screen.
- `x` An integer.
- `y` An integer.

**Values**

- `to-x` An integer.
- `to-y` An integer.

**Description**

The function `convert-relative-position` converts the position `x, y` in the coordinate system of `from` to that of `to`.

**Example**

See the example file `examples/capi/elements/convert-relative-position.lisp`.

**See also**

- `top-level-interface-geometry`
- `with-geometry`

---

**convert-to-screen**

**Function**

**Summary**

The `convert-to-screen` function finds the appropriate screen or container for a CAPI object.

**Package**

capi

**Signature**

```
convert-to-screen &optional object => result
```

**Arguments**

- `object` A CAPI object, a plist, or `nil`.

**Values**

- `result` A screen or a container.
This finds the appropriate screen or container for the CAPI object \textit{object}.

If \textit{object} is \texttt{nil}, \textit{result} is the default screen. \textit{object} defaults to \texttt{nil}.

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

\textit{object} can be a keyword representing the CAPI library. This is equivalent to using the \texttt{:library} key in the plist case below.

\textit{object} can be a plist. The keys below are supported on GTK+ and Motif. Other libraries ignore them.

\begin{description}
\item[\texttt{: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 \texttt{DISPLAY} environment variable or (on Motif) the \texttt{-display} command-line option, or (on GTK+) the \texttt{--display} command-line option. If neither is supplied, the default is to use the default screen on the local host.
\item[\texttt{:host}] The name of the host to use for the X Window System display. This key is valid only if no \texttt{:display} key/value is supplied. The default value is the local host.
\item[\texttt{:server-number}] The number of the display server to use for the X Window System display. This key is valid only if no \texttt{:display} key/value is supplied. The default value is 0.
\item[\texttt{:screen-number}] The number of the screen to use for the X Window System display. This key is valid only if no \texttt{:display} key/value is supplied. The default value is the default screen of the display.
\end{description}
: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.

```lisp
(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.

```lisp
(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: capi

Signature: current-dialog-handle => handle

Values: 
- handle: A platform-specific value, or nil.

Description: The function `current-dialog-handle` returns the underlying handle of the current dialog, as follows:

- **Microsoft Windows**: The hwnd of the dialog.
- **GTK+**: A pointer to the GdkWindow.
- **Motif**: A windowid of the dialog.
- **Cocoa**: The value returned by the NSWindow’s `windowNumber` method.

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

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

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

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

See also: `simple-pane-handle`
**current-document**

*Generic Function*

**Summary**
Returns the current document of a MDI interface.

**Package**
capi

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

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

**Values**
child The current document of mdi-interface.

**Description**
The generic function current-document returns the top child interface of a MDI interface.

**See also**
document-frame

---

**current-pointer-position**

*Function*

**Summary**
Returns the current position of the pointer.

**Package**
capi

**Signature**
current-pointer-position &key relative-to pane-relative-p => x, y

**Arguments**
relative-to A screen or a displayed interface or a CAPI pane.
pane-relative-p A boolean.

**Results**
x An integer.
y 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.

<table>
<thead>
<tr>
<th><strong>Signature</strong></th>
<th><code>current-popup =&gt; result</code></th>
</tr>
</thead>
</table>

- **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
`capi`

Signature
`current-printer &key interactive => printer`

Arguments
`interactive` A boolean.

Values
`printer` A printer, or `nil`.

Description
The `current-printer` function returns the currently selected printer object for the default library.

If `interactive` is non-nil and there is no current printer, a confirmmer is displayed warning the user and `printer` is `nil`. The default value of `interactive` is `nil`.

See also
`page-setup-dialog`
`set-printer-options`

**default-editor-pane-line-wrap-marker**

*Variable*

Summary
The default line wrap marker for editor panes.

Package
`capi`

Initial Value
`#\!`

Description
The variable `default-editor-pane-line-wrap-marker` provides the default value for the `line-wrap-marker` of an `editor-pane`. The value should be a `character` object, or `nil`.

See also
`editor-pane`
**default-library**  
*Function*

**Summary**
Returns the default library.

**Package**
capi

**Signature**
default-library => library

**Values**
library A library name.

**Description**
The function default-library returns a keyword naming the default library.

On Linux, FreeBSD and x86/x64 Solaris platforms, the default library is :gtk. If you load the deprecated "capi-motif" module, then the library will be :motif.

On Microsoft Windows platforms, currently the only library available is :win32, hence this is the default library.

On Mac OS X platforms, the only library available in the native GUI image is :cocoa, hence this is the default library. In the Mac OS X/GTK+ image, the default library is :gtk, but you load the deprecated "capi-motif" module, then the library will be :motif.

In LispWorks for UNIX only (not LispWorks for Linux, FreeBSD, or x86/x64 Solaris) platforms, currently the only library available is :motif, hence this is the default library.

**See also**
installed-libraries

---

**define-command**  
*Macro*

**Summary**
The define-command macro defines an alias for a mouse or keyboard gesture that can be used in the input model of an output pane.
C API Reference Entries

Package: capi

Signature: define-command name gesture &key translator host

Description:
The macro `define-command` defines an alias for a mouse or keyboard gesture that can then be used in `output-pane`'s input models. The `name` is the name of the alias and the `gesture` is one of the gestures accepted by `output-pane`. The `translator` is a function that gets passed the arguments that would be passed to the callback, and returns a list of arguments to be passed to the callback along with the `output-pane` (which will be the first argument). The `host` indicates which platforms this gesture should apply for (it defaults to all platforms).

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

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

```lisp
(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.

```lisp
(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
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:

- `:panes` Descriptions of the interface’s panes.
- `:layouts` Descriptions of the interface’s layouts.
- `:menus` Descriptions of the interface’s menus.
- `:menu-bar` A list of menus for the interface’s menu bar.
- `:definition` Options to alter `define-interface`.

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:

```lisp
(: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

```lisp
(: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

```lisp
(: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)
```

\[ key-spec := \text{var} | (\text{var}) | (\text{var initform}) | ((\text{keyword-name} \text{var})) | ((\text{keyword-name} \text{var}) \text{initform}) \]

\[ \text{keyword-name} := \text{any keyword} \]

`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.
Firstly, a couple of pane examples:

```lisp
(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:

```lisp
(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 '+ 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*)


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


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

```lisp
(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**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>sink</strong></td>
<td>A class instance.</td>
</tr>
<tr>
<td><strong>pane</strong></td>
<td>An <strong>ole-control-pane</strong>.</td>
</tr>
<tr>
<td><strong>interface-name</strong></td>
<td>A refguid or the symbol :default.</td>
</tr>
</tbody>
</table>

**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**

- interface: A CAPI interface.
- screen: A screen, or any argument accepted by convert-to-screen.
- owner: A CAPI interface.
- window-styles: A list of keywords.
- process: On GTK+, Windows or Motif, a CAPI process, t or nil. 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
### display-dialog

**Function**

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

**Package**
capi

**Signature**

```lisp
(display-dialog interface &key screen focus modal owner x y position-relative-to continuation callback-error-handler => result, okp)
```

**Arguments**

- `interface` A CAPI interface.
- `screen` A screen.
- `focus` A pane of `interface`.
- `modal` `t`, :dismiss-on-input or `nil`.
- `owner` A pane.
- `x`, `y` Real numbers representing coordinates, or keywords or lists specifying an adjusted position.
  - `position-relative-to` :
    - `owner` or `nil`.
- `continuation` A function or `nil`.
- `callback-error-handler` A function designator or `nil`.

**Values**

- `result` An object.
- `okp` A boolean.
Description

This is a complementary function that displays the CAPI 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 \texttt{abort-dialog}, causing the values \texttt{nil}, \texttt{nil} to be returned or to \texttt{exit-dialog} causing the values \texttt{result}, \texttt{t} to be returned, where \texttt{result} is the argument to \texttt{exit-dialog}. If \texttt{continuation} is non-nil, then the returned values are always \texttt{:continuation, nil}.

The CAPI also provides \texttt{popup-confirm} which gives you the standard \texttt{OK} and \texttt{Cancel} button functionality.

\texttt{callback-error-handler} allows error handling in callbacks which is uniform across platforms, as described for \texttt{popup-confirm}.

\textbf{Notes}

1. If you need to replace one dialog with another, you can use \texttt{display-replacable-dialog} and \texttt{replace-dialog}.

2. In a modal dialog at least one button which aborts or exits the dialog must be provided in \texttt{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 \texttt{popup-confirm} which adds the buttons for you.

\textbf{Example}

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

There are further examples in the directory \texttt{examples/capi/dialogs/}.

\textbf{See also}

\texttt{abort-dialog}
\texttt{display}
\texttt{display-replacable-dialog}
\texttt{exit-dialog}
\texttt{interface}
display-errors

Summary Displays a message if an error is signalled.

Package capi

Signature `display-errors &body body`

Description The macro `display-errors` executes the code of `body` inside a `handler-case` form. If an error is signalled inside `body`, a message is displayed and the debugger is not entered.

display-message

Summary The function `display-message` displays a message on the current CAPI screen.

Package capi

Signature `display-message format-string &rest format-args`

Description The function `display-message` creates a message from the arguments using `format`, 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 `prompt-with-message`.

Example

```lisp
(capi:display-message "Current screen = ~S"
 (capi:convert-to-screen))
```
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.
Package: capi

Superclasses: titled-object
               simple-pane

Initargs: :text
           A string or a list of strings to be displayed.

Accessors: display-pane-text

Description: The text passed to a display pane can be provided either as a single string containing newlines, or else as a list of strings where each string represents a line.

There are several classes which can display text, as follows:

title-pane
       Displays a single line of text.

display-pane
       Displays multiple lines of text.

text-input-pane
       Inputs a single line of text.

editor-pane
       Inputs multiple lines of text.

Example:

(capi:contain (make-instance
               'capi:display-pane
               :text
               '("One" "Line" "At" "A" "Time...")))

(setq dp (capi:contain (make-instance
                        'capi:display-pane
                        :text
                        '("One" "Line" "At" "A" "Time...")
                        :visible-min-height
                        '(:character 5))))

(capi:apply-in-pane-process
 dp #'(setf capi:display-pane-text
          '("Some" "New" "Text") dp))
See also  
display-pane
editor-pane
text-input-pane
title-pane

display-popup-menu  
Function

Summary  Displays a popup menu.

Package  capi

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

Arguments  

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

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

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

Alternatively the user may cancel the menu, by clicking elsewhere or pressing the Escape key. In this case, display-popup-menu returns nil.
owner specifies the owner of the menu, that is, a pane that the menu is associated with. If owner is not supplied the system tries to find the appropriate owner, which usually suffices.

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

button defaults to :button-3.

Example

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

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

See also

menu
pinboard-layout

display-replacable-dialog

Function

Summary
Displays a replacable dialog.

Package
capi

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

Arguments
interface An interface.
args Other arguments as for display-dialog.

Values
result The value returned by the dialog.

Description
The function display-replacable-dialog displays a dialog that can be replaced by another dialog.
*interface* is a CAPI interface to be displayed as a dialog. The arguments *args* are interpreted the same as the arguments to *display-dialog*, except that *modal* is ignored. *display-replacable-dialog* displays the dialog like *display-dialog*.

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

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

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

See also

- abort-dialog
- display-dialog
- exit-dialog
- replace-dialog

### display-tooltip

*Generic Function*

**Package**

capi

**Signature**

display-tooltip output-pane &key x y text => result

**Arguments**

- **output-pane**
  
  An instance of a subclass of output-pane.

- **x**
  
  The horizontal coordinate of the tooltip position.
The vertical coordinate of the tooltip position.

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-layout`s 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
docking-layout-controller
docking-layout-divider-p
docking-layout-docking-test-function
docking-layout-items

Readers
docking-layout-orientation

Description
The class docking-layout defines a region in which panes can be docked and undocked. The undocking functionality works only in LispWorks for Windows.

If controller is non-nil, it must be a controller object as returned by a call to make-docking-layout-controller. In this case the docking-layout is one of a group of docking-layouts 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 controller is nil (the default value), the docking-layout is in a Docking Group of one.

A pane pane 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 items passed to make-instance for some member of the group, or it has been set in some member by (setf docking-layout-items). The user can dock and undock pane in any member of the Docking Group. You can change the dockable status of panes programmatically by (setf docking-layout-items). You can query a pane’s docked and visible status in a docking-layout by docking-layout-pane-docked-p and docking-layout-pane-visible-p. You can change a pane’s docked and visible status in a docking-layout by (setf docking-layout-pane-docked-p) and (setf docking-layout-pane-visible-p).

By default, the context menu allows the user to alter the visibility status of each of the panes in the Docking Group.
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:

:title 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.

:docked-p A boolean specifying whether the pane should be docked. The default value is t. When a pane is not docked and is visible, it is displayed in its own window.

:visible-p A boolean specifying whether the pane is visible. The default value is t.
: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**

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

### docking-layout-pane-visible-p

**Function**

<table>
<thead>
<tr>
<th>Package</th>
<th>capi</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signature</td>
<td>docking-layout-pane-visible-p docking-layout pane =&gt; visiblep</td>
</tr>
</tbody>
</table>

107
**Signature**

\[
\text{(setf docking-layout-pane-visible-p) visiblep docking-layout pane => visiblep)}
\]

**Arguments**

- `docking-layout` An instance of `docking-layout` or a subclass.
- `pane` A pane.

**Values**

- `visiblep` A boolean.

**Description**

The function `docking-layout-pane-visible-p` returns a boolean indicating whether `pane` is currently visible in the Docking Group of `docking-layout`. `pane` may be docked in any member of the Docking Group, or undocked.

\[
\text{(setf docking-layout-pane-visible-p) may be used to change the visibility of pane in docking-layout only when pane is dockable in the Docking Group of docking-layout.}
\]

**See also**

- `docking-layout`

---

**document-container
document-container**

**Class**

**Package**

`capi`

**Superclasses**

`capi-object`

**Readers**

`screen-interfaces`

**Description**

The class of the container in a `document-frame`.

A document container has some screen-like functionality, responding to `screen-internal-geometry` and `screen-active-interface`.

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.

```lisp
(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.

```lisp
(capi:define-interface my-apropos-browser
  (capi:document-frame)
  ((string :initarg :string))
  (:panes
   (package-list capi:list-panel
    :items
    (loop for package in (list-all-packages)
       when
       (let ((al (apropos-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

(defun *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
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

```lisp
(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*

<table>
<thead>
<tr>
<th>Summary</th>
<th>Initiates a dragging operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Package</td>
<td>capi</td>
</tr>
<tr>
<td>Signature</td>
<td><strong>drag-pane-object</strong> pane value &amp;key string plist image-function operations =&gt; operation</td>
</tr>
<tr>
<td>Arguments</td>
<td>pane</td>
</tr>
<tr>
<td></td>
<td>value</td>
</tr>
<tr>
<td></td>
<td>string</td>
</tr>
<tr>
<td></td>
<td>plist</td>
</tr>
<tr>
<td></td>
<td>image-function</td>
</tr>
<tr>
<td></td>
<td>operations</td>
</tr>
<tr>
<td>Values</td>
<td>operation</td>
</tr>
<tr>
<td>Description</td>
<td>The function <strong>drag-pane-object</strong> initiates a dragging operation from within the pane <em>pane</em>. It can only be called from within the button :press or button :motion callbacks of the <em>input-model</em> of an <em>output-pane</em>. The <em>value</em>, <em>string</em> and <em>plist</em> arguments are combined to provide an object to be dragged in various formats. <em>value</em> can be any Lisp object (not necessarily a string) to make available for dropping into a pane within the local Lisp image. <em>string</em> can be a string representation of <em>value</em> to make available, or nill. If <em>string</em> is nill and <em>value</em> is a string, then that will be made available as the string.</td>
</tr>
</tbody>
</table>
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  
\begin{itemize}
  \item \textit{pane} \hspace{1em} An output-pane.
  \item \textit{metafile} \hspace{1em} A metafile, as described in \texttt{with-internal-metafile}.
  \item \textit{x,y} \hspace{1em} Integers.
  \item \textit{width,height} \hspace{1em} Non-negative integers.
\end{itemize}

Description  
The function \texttt{draw-metafile} draws the metafile \textit{metafile} to the pane \textit{pane} at position \textit{x,y} with size \textit{width}, \textit{height}.

\textit{metafile} should be a metafile as returned by \texttt{with-internal-metafile}.

\texttt{draw-metafile} is not implemented on GTK+ or X11/Motif.

Examples  
There is an example in \texttt{examples/capi/graphics/metafile.lisp}.

See also  
\begin{itemize}
  \item \texttt{clipboard}
  \item \texttt{draw-metafile-to-image}
  \item \texttt{free-metafile}
  \item \texttt{with-internal-metafile}
\end{itemize}
**draw-metafile-to-image**

**Function**

**Summary**
Draws a metafile as an image.

**Package**
capi

**Signature**
draw-metafile-to-image pane metafile &key width height max-width max-height background alpha => image

**Arguments**
- **pane**
  An output-pane.
- **metafile**
  A metafile.
- **width, height**
  Non-negative integers, or nil.
- **max-width, max-height**
  Non-negative integers, or nil.
- **background**
  A color specification.
- **alpha**
  A generalized boolean.

**Values**
- **image**
  An image.

**Description**
The function **draw-metafile-to-image** returns a new image object for pane, with metafile drawn into the image.

**metafile** should be a metafile as returned by with-internal-metafile.

If **width** and **height** are both nil then the size of the image is computed from the metafile. If both **width** and **height** are integers, then they specify the size of the image and the metafile is scaled to fit. If one of **width** or **height** is nil, then it is computed from the other dimension, preserving the aspect ratio of the metafile. The default values of **width** and **height** are both nil.

The **max-width** and **max-height** arguments, if non-nil, constrain the computed or specified values of **width** and **height** respectively. The aspect ratio is retained when the size is con-
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
The display-callback is called with the output pane to draw on, the drawn-pinboard-object itself, and the x, y, width and height of the object, and it is expected to redraw that section.

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

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

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

See also pinboard-layout

draw-pinboard-object

Generic Function

Summary
Draws a pinboard object.

Package
capi

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

Description
The generic function draw-pinboard-object is called whenever a pinboard object needs to be drawn. The x, y, width and height arguments indicate the region that needs to be redrawn, but a method is free to ignore these and draw the complete object.
Example
See the example in the file
examples/capi/graphics/circled-graph-nodes.lisp

See also
pinboard-layout
pinboard-object

draw-pinboard-object-highlighted  

Generic Function

Summary  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 `draw-pinboard-object-unhighlighted` removes the highlighting from a pinboard object.

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

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

---

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

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

**Package**  
capi

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

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

**Values**  
- `result`  
  A boolean

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

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

**See also**  
`drop-object-drop-effect`
`simple-pane`
**drop-object-collection-index**

*Function*

**Summary**

Gets the index and relative place in the *collection* that an object is being dropped over.

**Signature**

```
drop-object-collection-index drop-object => index, placement
```

```
(setf (drop-object-collection-index drop-object) (values
  new-index new-placement))
```

**Arguments**

`drop-object`

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

`new-index`

An integer.

`new-placement`

One of :above, :item or :below.

**Values**

`index`

An integer.

`placement`

One of :above, :item or :below.

**Description**

The function *drop-object-collection-index* returns the index and place relative to that index within the *collection* that the object *drop-object* is being dropped over. This information is only meaningful when the pane is an instance of list-panel or tree-view.

The returned value `index` is the position in the *collection* (see `get-collection-item` or `choice-selection`). The returned value `placement` indicates whether the user is dropping above, on or below the item at `index`.

There is also a `setf` expander that can be called with these two values within the :drag stage of the operation, to adjust where the user will be allowed to drop the object.

**Notes**

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

For an example illustrating the use of drag and drop in a choice, see
examples/capi/choices/drag-and-drop.lisp

See also
drop-object-collection-item

drop-object-collection-item  

Function

Summary

Gets the item and relative place in the collection that an object is being dropped over.

Signature

drop-object-collection-item drop-object  => item, placement

(setf (drop-object-collection-item drop-object) (values new-item new-placement))

Arguments

drop-object  A drop-object, as passed to the drop-callback.
new-item     An item of a collection.
new-placement One of :above, :item or :below.

Values

item          An item of a collection.
placement     One of :above, :item or :below.

Description

The function drop-object-collection-item returns the item and place relative to that item within the collection that the object drop-object is being dropped over. This information is only meaningful when the pane is an instance of list-panel or tree-view.

The returned value placement indicates whether the user is dropping above, on or below the item.

There is also a setf expander that can be called with these two values within the :drag stage of the operation, to adjust where the user will be allowed to drop the object.
drop-object-collection-item should only be called within a drop-callback. It is not supported on X11/Motif. See simple-pane for information about drop callbacks.

For an example illustrating the use of drag and drop in a choice, see examples/capi/choices/drag-and-drop.lisp

See also drop-object-collection-index

drop-object-drop-effect

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

Package
capi

Signature
drop-object-drop-effect drop-object => effect

Signature
(setf drop-object-drop-effect) effect drop-object => effect

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

Values
effect An effect keyword

Description
The function drop-object-drop-effect gets or sets the current effect of the dropping operation. effect can be one of:

:copy The object will be copied. This is the most common value for operations between applications.

:move The object will be moved. This is usually triggered by the user dragging with a platform-specific modifier key pressed.

:link A link to the object will be created. This is usually triggered by the user dragging with a platform-specific modifier key pressed.
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**
capi

**Signature**

\[
\text{drop-object-pane-x drop-object} \Rightarrow x\text{-coord} \\
\text{drop-object-pane-y drop-object} \Rightarrow y\text{-coord}
\]

**Arguments**

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

**Values**

*x-coord, y-coord* Integers.

**Description**
The functions *drop-object-pane-x* and *drop-object-pane-y* 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 *output-pane* or one of its subclasses.

**Notes**

*drop-object-pane-x* and *drop-object-pane-y* should only be called within a *drop-callback*. They are not supported on X11/Motif. See *simple-pane* for information about drop callbacks.

**See also**
simple-pane

---

**drop-object-provides-format**

*Function*

**Summary**
Queries whether a dropping operation can provide an object in a given format.

**Package**
capi

**Signature**

\[
\text{drop-object-provides-format drop-object format} \Rightarrow \text{result}
\]
Arguments  drop-object  A drop-object, as passed to the drop-callback.

format  A format keyword

Values  result  A boolean

Description  The function drop-object-provides-format returns non-nil if the dropping operation can provide an object in the given format. It returns nil if it cannot provide that format.

See set-drop-object-supported-formats for information on format keywords.

Note: drop-object-provides-format 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

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

Summary  The drawing style of the Echo Area cursor when the window is inactive.

Package  capi

Initial Value  :invisible

Description  The drawing style of the cursor in the Echo Area of an inactive window in the LispWorks IDE.

The allowed values are :inverse, :outline, :underline and :invisible.
echo-area-pane

Class

Summary The class of the Editor’s echo area.

Package capi

Superclasses editor-pane

Description The class echo-area-pane 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.

You should not normally need to work with this class directly. To add an Echo Area, pass :echo-area t when making the editor-pane.

*editor-cursor-color*

Variable

Summary The background color of the cursor.

Package capi

Initial Value nil

Description When non-nil, the value is a color spec or color alias determining the background color of the editor-pane cursor. See "The Color System" in the LispWorks CAPI User Guide for information about colors in LispWorks.

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

Example (setf capi:*editor-cursor-color* :red)
*editor-cursor-active-style*  
Variable

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

Package  capi

Initial Value  :inverse

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

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

See also  editor-pane-blink-rate

*editor-cursor-drag-style*  
Variable

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

Package  capi

Initial Value  :left-bar

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

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

*editor-cursor-inactive-style*  
Variable

Summary  The drawing style of the editor’s cursor when the window is inactive.
Package: capi

Initial value: :outline

Description: The drawing style of an editor-pane cursor when the window is inactive.

The allowed values are :inverse, :outline, :underline or :invisible.

**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:

: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.
:echo-area      A flag determining whether the editor pane has an Echo Area.
:fixed-fill     An integer specifying the fill length, or nil.
:line-wrap-marker
                A character, or nil.
:line-wrap-face
                An editor:face object, or a symbol naming a face, or nil.
:wrap-style     An integer specifying the fill length, or nil.

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:

- t: Normal wrapping. Display as many characters as possible in the editor-pane line.
- nil: Do not wrap. Text lines that are too long are truncated.
:split-on-space

Wrapping, but attempts to split lines on spaces. When the text reaches the end of a line, the code looks backwards for space, and wraps before it.

The default value of wrap-style is t and the value can 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:

```lisp
(capi:define-interface updating-editor ()
 ()
 (:panes 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 self => blink-rate

Arguments  self  An editor pane.

Values  blink-rate  A non-negative real number, or nil.

Description  The system calls the function editor-pane-blink-rate to determine the cursor blinking rate in milliseconds. The pane uses the value blink-rate 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**
capi

**Signature**
`editor-pane-buffer pane`

**Description**
The function `editor-pane-buffer` 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**
```lisp
(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..."))
```

See also
* `editor-pane`
editor-pane-native-blink-rate

**Function**

**Summary**
Returns the native cursor blinking rate for an editor-pane.

**Package**
capi

**Signature**
editor-pane-native-blink-rate pane => blink-rate

**Arguments**
pane An editor-pane.

**Values**
blink-rate A non-negative real number, or nil.

**Description**
The function `editor-pane-native-blink-rate` returns the native cursor blinking rate for the editor-pane pane, that is the rate that the GUI library (Motif, Microsoft Windows, Cocoa) uses.

The value `blink-rate` is interpreted as a blinking rate as described in `editor-pane-blink-rate`.

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

editor-pane-selected-text

**Function**

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

**Package**
capi

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

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

**Values**
result A string or nil.
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  

editor-pane  

An editor-pane.

Values  

stream  

An output stream.

Description  

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

See also  

editor-pane

editor-window  

Generic Function

Summary  

Returns the editor window object.

Package  

capi

Signature  

editor-window editor => editor-window

Arguments  

editor  

An editor-pane or an Editor interface in the LispWorks IDE.

Values  

editor-window  

An editor window object.

Description  

The generic function editor-window returns the editor window object associated with editor.

The functionality of editor windows is documented in the LispWorks Editor User Guide.

See also  

editor-pane

element  

Class

Summary  

The class element is the superclass of all CAPI objects that appear in a window.
Initargs

:parent  The element containing this element.
:interface  The interface containing this element.
:accepts-focus-p  Specifies that the element should accept input.
:help-key  An object used for lookup of help. Default value t.
:widget-name  A string designator.

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

:x  The $x$ position of the element in a pinboard.
:y  The $y$ position of the element in a pinboard.
:external-min-width  The minimum width of the element in its parent.
:external-min-height  The minimum height of the element in its parent.
:external-max-width  The maximum width of the element in its parent.
:external-max-height  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 `cl: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:

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

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

- `max` The maximum size of the hints.
- `min` The minimum size of the hints.
- `+` The sum of the hints.
- `-` The subtraction of hints from the first.
- `*` The multiplication of the hints.
- `/` 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:
The size of `integer` characters.

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

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

```lisp
(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.

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

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

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

```lisp
(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  capi

Signature  element-container element => container

Arguments  element  An element.

Values  container  A screen or a document-frame.

Description  The function element-container returns the container of the element element.

If element is inside a standalone interface, then container is the screen object.

If element is inside an interface that is inside a MDI interface, then container is the capi:container object of that MDI interface. See document-frame for details.

See also  document-frame  element

**element-interface-for-callback** *Function*

Summary  Returns the interface that is used in an element’s callbacks.

Package  capi

Signature  element-interface-for-callback element => interface

Description  The function element-interface-for-callback returns the interface that is passed to callbacks in element. Normally this is the interface that element is in, but that can be changed by attach-interface-for-callback.
element-screen  

Function

Summary  Returns the screen that an element is associated with.

Package  capi

Signature  \texttt{element-screen element => screen}

Description  The function \texttt{element-screen} returns the screen that the element \texttt{element} is associated with.

See also  \texttt{element}

ellipse  

Class

Summary  A pinboard object that draws itself as an ellipse.

Package  capi

Superclasses  \texttt{pinboard-object}

Accessors  \texttt{filled}

Initargs  \texttt{:filled}  A boolean.

Description  The class \texttt{ellipse} is a \texttt{pinboard-object} that draws itself as an ellipse.

If \texttt{filled} is true, then the ellipse is filled with the foreground color. \texttt{filled} defaults to \texttt{nil}. 

See also  \texttt{attach-interface-for-callback, element}
**ensure-area-visible**  
*Generic Function*

**Summary**
Ensures an area is visible in a scrollable pane.

**Package**
capi

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

**Arguments**
- self: A simple-pane with internal scrolling.
- x, y: The coordinates of the origin of the area to make visible.
- width, height: The dimensions of the area to make visible.

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

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

**ensure-interface-screen**  
*Function*

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

**Package**
capi

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

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

execute-with-interface

Function

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

Package
capi

Signature
execute-with-interface interface function &rest args

Arguments
interface An interface
function A function designator
args Arguments passed to function

Description
The function execute-with-interface is a useful way of operating on an interface 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
1. execute-with-interface applies function even if interface does not have a screen representation, for example when it is destroyed. To call function only if interface has a representation, use execute-with-interface-if-alive.

2. 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 execute-with-interface is a useful utility in other circumstances.
3. **execute-with-interface** calls *function* on the current process if *interface* does not have a process.

**Example**

```lisp
(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**

```lisp
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.
exit-confirm

Function

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

Package
capi

Signature
exit-confirm &rest dummy-args

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

Example
This example demonstrates the use of exit-confirm 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-confirm.

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

See also
popup-confirm
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-confirm
interface

expandable-item-pinboard-object

Class

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

Package: capi

Superclasses: item-pinboard-object
### expandable-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  

row-layout

Initargs

:callback-object

The argument for the callbacks. If it is nil the top-level-interface of the layout is used.

:change-callback

A function of one argument (the callback-object). It is called whenever the text in the filter changes. Also if callback is not supplied, change-callback is called instead.

:callback

A function of one argument (the callback-object). It is called when the user presses Return, makes a selection from the menu, or clicks the Confirm button. If callback is not supplied, change-callback is called instead.

:text

A string specifying the initial text of the filter, or nil.

:matches-title

A string, t or nil.

:help-string

A string, t or nil.

:label-style

:short, :medium or :long.

Accessors

filtering-layout-state
filtering-layout-matches-text

Description

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

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

The filtering layout defines the parameters to use, and calls the callbacks to perform the filtering. It does not do any filtering itself.

To actually do the filtering, the using code needs to call `filtering-layout-match-object-and-exclude-p`, which returns as multiple values a precompiled regexp and a flag specifying whether to exclude matches. The regexp should be used to perform the filtering, typically by using `lisp-works:find-regexp-in-string`. Note that `filtering-layout-match-object-and-exclude-p` returns `nil` when there is no string in the `text-input-pane`, and that even when the filter is set to plain match it is returns a regexp (which matches a plain string).

You supply a filtering-layout amongst the panes of your interface definition (not its layouts). The description of a filtering-layout is set by the `initialize-instance` method of the class, and therefore the description cannot be passed as an initarg and should not be manipulated.

`filtering-layout-state` returns a "state" object which can be used later to set the state of any filtering-layout by `(setf capi:filtering-layout-state)`. When setting the state, the value can also be a string or `nil`. A string means setting the filter string to it and making the filtering state be plain string, includes matches, and case-insensitive. `nil` means the same as the empty string.

`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 t, 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

```lisp
(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 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 `filtering-layout`.

**Package**
capi

**Signature**
`filtering-layout-match-object-and-exclude-p` filtering-layout display-message => regexp, excludep

**Arguments**
- filtering-layout: A `filtering-layout`
- display-message: A generalized boolean

**Values**
- regexp: A precompiled regular expression
- excludep: A boolean

**Description**
The function `filtering-layout-match-object-and-exclude-p` returns a regexp to use for filtering in the `filtering-layout`. The second returned value `excludep` specifies whether the filter should be used to exclude or include matches.

`display-message` is a generalised boolean controlling whether a message is displayed to the user if there is an error when compiling the regexp.

See `filtering-layout` for details.

**See also**
- `filtering-layout`

---

find-graph-edge

**Generic Function**

**Summary**
Finds and returns an edge in a graph given two items.

**Package**
capi

**Signature**
`find-graph-edge` graph from to => edge

**Arguments**
- graph: A `graph-pane`. 
from       An item in graph.
to         An item in graph.

Values     edge       A graph edge, or nil.

Description The generic function find-graph-edge finds the edge that
goes from the node corresponding to from to the node correspond-
sing to to.
If there is no such edge, find-graph-edge returns nil.

See also   find-graph-node
            graph-pane

find-graph-node

Generic Function

Summary    Finds and returns a node in a graph corresponding to an
            item.

Package    capi

Signature  find-graph-node graph object => node

Arguments  graph       A graph-pane.
            object      An item in graph.

Values     node       A node of graph, or nil.

Description The generic function find-graph-node finds the node that
corresponds to the item object.
If there is no such node, find-graph-node returns nil.

See also   find-graph-edge
            graph-pane
find-interface

Generic Function

Summary
Displays an interface of a given class, making it if necessary.

Package
capi

Signature
find-interface class-name &rest initargs &key screen &allow-other-keys => interface

Arguments
- **class-name**: A specifier for a subclass of interface.
- **initargs**: Initialization arguments for class-name.
- **screen**: A screen or nil.

Values
- **interface**: An interface of class class-name.

Description
The generic function find-interface finds and displays an interface of the given class class-name that matches initargs and screen.

class-name can be the name of a suitable class, the class itself, or an instance of the class.

screen can be a CAPI object as accepted by convert-to-screen. screen defaults to the default screen.

find-interface calls locate-interface to locate an existing interface:

1. If an interface of the class specified by class-name matching initargs exists already on screen, then this interface is activated and returned.

2. Otherwise, if an interface of the class specified by class-name exists already on screen, then reinitialize-interface is applied to this interface which is then activated and returned.

If no instance of class class-name exists on screen, then find-interface creates one by passing class-name and initargs to make-instance, and displays the result on screen.
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`
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

Summary
The class form-layout lays its children out in a form.

Package
capi

Superclasses
layout

Initargs
:vertical-gap The gap between rows in the form.
:vertical-adjust
The adjustment made to the rows.
:title-gap
The gap between the two columns.
:title-adjust
The adjustment made to the left column.

Accessors
form-vertical-gap
form-vertical-adjust
form-title-gap
form-title-adjust

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
This class has been superseded by grid-layout, and will
probably be removed at some point in the future. The exam-
pies below demonstrate the use of grid layouts as an alterna-
tive to forms.

Example
(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))))
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/metafile.lisp`.

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

free-sound

Function

Summary Frees a loaded sound object.
**get-collection-item**

*Generic Function*

**Summary**

Returns the item at a specified position in a collection.

**Package**

capi

**Signature**

generic-function get-collection-item self index

**Description**

The generic function get-collection-item returns the item at position index from the collection self. It achieves this by calling the items-get-function of the collection. There is also a complementary function, search-for-item which finds the index for a given item in a collection.

**See also**

collection

search-for-item

**get-constraints**

*Function*

**Summary**

Returns a list of the constraints for an element.

**Package**

capi
**Signature**

get-constraints element

**Description**

The function `get-constraints` returns the constraints for `element` as multiple values (the values are the minimum width, the minimum height, the maximum width and the maximum height).

This function calls the generic function `calculate-constraints` 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 `invalidate-pane-constraints`.

**See also**

- `calculate-constraints`
- `define-layout`
- `element`
- `invalidate-pane-constraints`

---

**get-horizontal-scroll-parameters**

*Generic Function*

**Summary**

Queries the scroll parameters of a horizontal scroll bar.

**Package**

capi

**Signature**

get-horizontal-scroll-parameters self &rest keys => parameter, parameter,...

**Arguments**

- `self` A displayed `simple-pane`.
- `keys` Keywords as below.

**Values**

- `parameter` The parameters are returned as multiple values, one for each key passed in `keys` and in the same order as the arguments.
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.

**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`
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` functions 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.
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**  
`get-vertical-scroll-parameters self &rest keys => parameter, parameter,...`

**Arguments**  
- `self`  
  A displayed `output-pane` or `layout`.
- `keys`  
  Keywords as below.

**Values**  
- `parameter`  
  The parameters are returned as multiple values, one for each key passed in `keys` and in the same order as the arguments.

**Description**  
The function `get-vertical-scroll-parameters` retrieves 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 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-pane`, the underlying widget decides what the scroll range and units are.

**Example**  
See the following CAPI example files:
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 capi

Subclasses graph-edge  
graph-node

Readers graph-object-element  
graph-object-object

Description The class graph-object is the superclass of graph-edge and graph-node.

The reader graph-object-element returns the CAPI object that is displayed.

The reader graph-object-object returns the user object associated with the graph object.

description

Class

Summary A graph pane is a pane that displays a hierarchy of items in a graph.

Package capi

Superclasses simple-pinboard-layout  
choice

Subclasses simple-network-pane

Initargs :roots The roots of the graph.

:children-function Returns the children of a node.

:layout-function A keyword denoting how to layout the nodes.
The adjust value for the x direction.

The adjust value for the y direction.

The class of pane to represent nodes.

The class of pane to represent edges.

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 of 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 \textit{adjust} are interpreted as by \texttt{pane-adjusted-position}. \texttt{:top} is the default for \textit{layout-y-adjust} and \texttt{:left} is the default for \textit{layout-x-adjust}.

When a graph pane wants to display nodes and edges, it creates instances of \texttt{node-pinboard-class} and \texttt{edge-pinboard-class} which default to \texttt{item-pinboard-object} and \texttt{line-pinboard-object} respectively. These classes must be subclasses of \texttt{simple-pane} or \texttt{pinboard-object}, and there are some examples of the use of these keywords below.

The \textit{node-pane-function} is called to create a pane for each node, and by default it creates an instance of \texttt{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 \texttt{simple-pane} or \texttt{pinboard-object}.

To expand or contract a node, the user clicks on the circle next to the node. An expandable node has an unfilled circle and a collapsable node has a filled circle.

\texttt{graph-pane} is a subclass of \texttt{choice}, so for details of its selection handling, see \texttt{choice}.

The highlighting of the children is controlled as described for \texttt{pinboard-layout}, but for \texttt{graph-pane} the default value of \textit{highlight-style} is \texttt{:standard}.

\textbf{Compatibility note} In LispWorks 4.3 the double click gesture on a \texttt{graph-pane} node always calls the \textit{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 \textit{action-callback} was not always called.

\textbf{Example}

\begin{verbatim}
(defun node-children (node)
  (when (< node 16)
    (list (* node 2)
          (1+ (* node 2)))))
\end{verbatim}
(setq graph
  (capi:contain 'capi:graph-pane
    :roots '(1)
    :children-function 'node-children
    :best-width 300 :best-height 400))

(capi:apply-in-pane-process
  graph #'(setf capi:graph-pane-roots) '(2 6) graph)

(capi:contain
  (make-instance 'capi:graph-pane
    :roots '(1)
    :children-function 'node-children
    :layout-function :top-down
    :best-width 300 :best-height 400)

(capi:contain
  (make-instance 'capi:graph-pane
    :roots '(1)
    :children-function 'node-children
    :layout-function :top-down
    :layout-x-adjust :left)
    :best-width 300 :best-height 400)

This example demonstrates a different style of graph output with right-angle edges and parent nodes being adjusted towards the top instead of at the center.

(capi:contain
  (make-instance 'capi:graph-pane
    :roots '(1)
    :children-function 'node-children
    :layout-y-adjust '(:top 10)
    :edge-pinboard-class 'capi:right-angle-line-pinboard-object)
    :best-width 300 :best-height 400)

This example demonstrates the use of
  :node-pinboard-class to specify that the nodes are drawn as push buttons.
(capi:contain
(make-instance
 'capi:graph-pane
 :roots '(1)
 :children-function 'node-children
 :node-pinboard-class 'capi:push-button)
 :best-width 300
 :best-height 400)

There are more examples in the directory
examples/capi/graphics/.

See also
item-pinboard-object
line-pinboard-object

**graph-pane-add-graph-node**

*Generic Function*

**Summary**
Adds a node to a graph.

**Package**
capi

**Signature**
`graph-pane-add-graph-node graph-pane object parent-node => new-node`

**Arguments**
- `graph-pane` A graph-pane.
- `object` An object.
- `parent-node` A graph-node.

**Values**
- `new-node` A graph-node.

**Description**
The generic function `graph-pane-add-graph-node` adds a new node in the graph `graph-pane` corresponding to `object`, and links it as a child of `parent-node`.

**See also**
graph-node
graph-pane
**graph-pane-delete-object**  
*Generic Function*

**Summary**  
Removes a node from a graph.

**Package**  
capi

**Signature**  
`graph-pane-delete-object graph-pane object`

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

**Description**  
The generic function `graph-pane-delete-object` deletes the node corresponding to `object` in the graph `graph-pane`.

**See also**  
- `graph-node`
- `graph-pane`
- `graph-pane-add-graph-node`
- `graph-pane-delete-objects`

---

**graph-pane-delete-objects**  
*Generic Function*

**Summary**  
Removes nodes from a graph.

**Package**  
capi

**Signature**  
`graph-pane-delete-objects graph-pane objects`

**Arguments**  
- `graph-pane`  
  A `graph-pane`.
- `objects`  
  A list of objects.

**Description**  
The generic function `graph-pane-delete-objects` deletes the node in the graph `graph-pane` corresponding to each object in the list `objects`.
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.
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

```lisp
(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**

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**

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`. 
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: capi

Signature: graph-pane-update-moved-objects graph-pane objects

Arguments:
- graph-pane: A graph-pane.
- objects: A list.

Description:
The generic function `graph-pane-update-moved-objects` is called after some objects in the graph `graph-pane` were moved by a user gesture. `objects` 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: graph-pane

grid-layout

Class

Summary: The `grid-layout` is a layout which positions its children on a two dimensional grid.

Package: capi

Superclasses: x-y-adjustable-layout

Subclasses: row-layout

:orientation

Initargs:
- :columns: The number of columns in the grid.
- :has-title-column-p: A boolean specifying whether the first column is a title column.
- :orientation: 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 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 \textit{x-ratios} (or \textit{y-ratios}) control the size of each child relative to the others. If an element in \textit{x-ratios} (or \textit{y-ratios}) is \texttt{nil} the child is fixed at its minimum size. Otherwise the size is calculated as follows

\[(\text{round} \ (* \ \text{total} \ \textit{ratio} \ \textit{ratio-sum})\)

where \textit{ratio-sum} is the sum of the non-nil elements of \textit{x-ratios} (or \textit{y-ratios}) and \textit{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 \textit{x-ratios} (or \textit{y-ratios}) has fewer elements than the number of children, 1 is used for each of the missing ratios. Leaving \textit{x-ratios} (or \textit{y-ratios}) \texttt{nil} causes all of the children to be the same size.

The positions of each pane in the layout can be specified using \textit{x-adjust} and \textit{y-adjust} like every other \texttt{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 \texttt{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 \texttt{:right-extend} (or \texttt{: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 \texttt{description} that you want the item to expand into. For \texttt{: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

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Example

Examp
(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)))

This example illustrates the special interpretation of nil in the description:

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

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

Summary  Hides the specified pane.
hide-pane pane => pane

Arguments

pane An instance of simple-pane or a subclass.

Description

The function hide-pane hides the pane pane, removing it from the screen. pane's children, if any, are hidden too.

To restore pane to the screen, use show-pane.

See also

hide-interface
show-pane

highlight-pinboard-object

Generic Function

Summary

Highlights a specified pinboard object.

Arguments

pinboard A pinboard-layout.
object A pinboard-object.
redisplay A generalised boolean.

Description

The generic function highlight-pinboard-object causes the pinboard object object to become highlighted until unhighlight-pinboard-object is called on it.

The pinboard object highlighting is drawn according to the highlight-style of the pinboard-layout pinboard.

If redisplay is non-nil the highlighting is drawn immediately.

The default value for redisplay is t.
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

- `examples/capi/choice/tree-view.lisp`
- `examples/capi/choice/extended-selection-tree-view.lisp`

**See also**

- `image-set`
- `load-image`
- `register-image-translation`

---

**image-pinboard-object**

**Class**

**Summary**

An image pinboard object is a pinboard object that displays itself as an image.

**Package**

capi

**Superclasses**

- `pinboard-object`
- `titled-object`

**Initargs**

- `:image` The image to be displayed.

**Accessors**

- `image-pinboard-object-image`

**Description**

The `image` initarg for an `image-pinboard-object` should either be an `external-image` or any other object accepted by `load-image`. The image displayed in the object can be changed dynamically using the writer function

```
(setf image-pinboard-object-image)
```
Example

```lisp
(cdd (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
name A string.
if-exists One of :supersede, :error or nil.
default One of t, nil or :when-none.
savep A boolean.
ppd-file A string or pathname.
description A string, or :preserve.
use-jcl A boolean, or :preserve.
command A string, or :preserve.
use-file A boolean, or :preserve.
always-print-to-file
A boolean, or :preserve.
orientation One of :landscape, :portrait or :preserve.
installed-options An association list, or :preserve.
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.
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.

This example assumes there is just one line of output from each command sent to the pipe

```lisp
(capi:contain
  (make-instance 'capi:interactive-pane
    :top-level-function #'(lambda (interface pane stream)
      (declare (ignore interface pane))
      (with-open-stream (s (sys:open-pipe '
("/usr/local/bin/bash")
        :direction :io))
        (loop
          (progn
            (format stream "primitive xterm$ ")
            (let ((input (read-line stream nil nil)))
              (if input
                (progn
                  (write-line input s)
                  (force-output s))
                (return)))))
            (let ((output (read-line s nil nil)))
              (if output
                (progn
                  (write-line output stream)
                  (force-output stream))
                (return))))))))
  :best-height 300
  :best-width 300)
```

See also

`collector-pane`
**interactive-pane-execute-command**

*Generic Function*

**Summary**
Simulates user entry of commands in an interactive-pane.

**Package**
capi

**Signature**
`interactive-pane-execute-command interactive-pane command
&key command-modification-function editp &allow-other-keys`

**Arguments**
- `interactive-pane` An interactive-pane.
- `command` A Lisp form.
- `command-modification-function` A function or nil.
- `editp` A generalized boolean.

**Description**
The generic function `interactive-pane-execute-command` has the same effect as the user typing the Lisp form `command` into the interactive-pane interactive-pane, and pressing Return.

`interactive-pane-execute-command` may be called from any process.

If `command-modification-function` is non-nil, it is a function of one argument. It is called with argument `command` in the process in which `interactive-pane` runs. The result of this call is used as the command to enter. The default value of `command-modification-function` is nil.

If `editp` is true then the command is left at the end of the pane for the user to edit before pressing Return. If `editp` is nil then `interactive-pane-execute-command` simulates the user pressing Return. The default value of `editp` is nil.

**See also**
- `interactive-pane`
- `listener-pane-insert-value`
The class `interface` is the top level window class, which contains both menus and a hierarchy of panes and layouts. Interfaces can also themselves be contained within a layout, in which case they appear without their menu bar.

**Package**
capi

**Superclasses**
simple-pane
titled-object

**Initargs**

- **:title** The title of the interface.
- **:layout** The layout of the interface.
- **:menu-bar-items** The items on the menu bar.
- **:auto-menus** A flag controlling the automatic addition of system menu objects.
- **:create-callback** A callback done on creating the window, before display and user interaction.
- **:destroy-callback** A callback done on closing the window.
- **:confirm-destroy-function** A function to verify closing of the window.
- **:best-x** The best x position for the interface.
- **:best-y** The best y position for the interface.
- **:best-width** The best width of the interface.
- **:best-height** 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-tooltip
   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

- 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-tooltip-enabled
- interface-help-callback
- top-level-interface-external-border
- top-level-interface-transparency
- interface-toolbar-items
- interface-toolbar-states
- interface-default-toolbar-states

Readers

- interface-window-styles

Description

Every interface can have a title *title* 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 *titled-object* for more details).

The argument *layout* specifies a layout object that contains the children of the interface. To change this layout you can either use the writer *pane-layout*, or you can use the layout *switchable-layout* which allows you to easily switch the currently visible child.

The argument *menu-bar-items* specifies a list of menus to appear on the interface’s menu bar.

*auto-menus* defaults to *t*, which means that an interface may have some automatic menus created by the environment in which it is running (for example the *Works* menu in the Lisp-Works IDE). To switch these automatic menus off, pass *:auto-menus nil*. 
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

inconify-callback may be nil, meaning there is no callback. This is the default value. Otherwise inconify-callback is a function of two arguments: the interface and a boolean iconify which is true when interface is iconified and and false when it is restored. Its signature is:

inconify-callback interface iconify

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 `eql` 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

```lisp
(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) nil))))
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.

```lisp
(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.

```lisp
(setf (capi:interface-override-cursor interface) :i-beam)
```

Remove the override cursor.

```lisp
(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))

219
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*

<table>
<thead>
<tr>
<th><strong>Summary</strong></th>
<th>The function called to display an interface on screen.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Package</strong></td>
<td>capi</td>
</tr>
</tbody>
</table>
| **Signature** | interface-display  
interface | |
| **Arguments** | interface  
An instance of a subclass of interface. | |
| **Description** | The generic function interface-display is called by display to display an interface on screen.  
The primary method for interface actually does the work.  
You can add :before methods on your own interface classes for code that needs to be executed just before the interface appears, and :after methods for code that needs to be executed just after the interface appears. | |
| **Notes** | 1. interface-display is called in the process of interface.  
2. interface-display is not called when interface is displayed as a dialog. Another way to run code before it appears on screen is to supply a create-callback for interface. | |
| **Example** | This example shows how interface-display can be used to set the initial selection in a choice whose items are computed at display-time: | |
(capi:define-interface my-tree ()
  ((favorite-color :initform :blue))
  (:panes
   (tree
capi:tree-view
   :roots '(:red :blue :green)
   :print-function
   'string-capitalise))
  (:default-initargs
   :width 200
   :height 200))

(defmethod capi:interface-display :after
  ((self my-tree))
  (with-slots (tree favorite-color) self
    (setf (capi:choice-selected-item tree)
      favorite-color)))

(capi:display (make-instance 'my-tree))

See also
display
interface

**interface-display-title**

**Function**

**Summary**
Returns the interface title to use on screen.

**Package**
capi

**Signature**
interface-display-title interface => string

**Arguments**
interface A CAPI interface.

**Values**
string A string.

**Description**
The function `interface-display-title` returns the title to use when displaying the interface `interface` on screen.

This is equivalent to:
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

interface  A CAPI interface.
title  A string.

Description

The generic function `interface-extend-title` is called by the system with an interface and its title before actually displaying the title on the screen. The result must be a string, which is actually displayed. There is no requirement for any relation between the title argument and the result.

The return value `string` is the title to display on the screen.

The default method uses the values set by `set-default-interface-prefix-suffix`. You can specialize `interface-extend-title` to get other effects.

See also

`interface-display-title`

`set-default-interface-prefix-suffix`

### `interface-geometry`  

**Generic Function**

**Summary**  

Returns the geometry of an interface.

**Package**  

capi

**Signature**  

`interface-geometry interface => geometry`

**Arguments**  

interface  An instance of a subclass of `interface`.

**Values**  

geometry  A list.

**Description**  

The generic function `interface-geometry` returns a list representing the geometry of interface in pixel values. `geometry` is of the form `(x y width height)`.

See also  

`interface`
**interface-iconified-p**

*Function*

**Summary**
The predicate for whether an interface is iconified.

**Package**
capi

**Signature**
`interface-iconified-p pane => iconifiedp`

**Arguments**
`pane` A CAPI element.

**Values**
`iconifiedp` A boolean.

**Description**
The function `interface-iconified-p` returns `t` if the top level interface containing `pane` is iconified. This means that the window is visible as an icon, also referred to as minimized.

If the top level interface is not iconified, then `interface-iconified-p` returns `nil`.

**See also**
hide-interface
top-level-interface
top-level-interface-display-state

---

**interface-keys-style**

*Generic Function*

**Summary**
Determines the emulation for an interface.

**Package**
capi

**Signature**
`interface-keys-style interface => keys-style`

**Arguments**
`interface` An instance of a subclass of `interface`.

**Values**
`keys-style` A keyword, :pc, :emacs or :mac.
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 Alt key on Microsoft Windows, selected text, and accelerators:

- **:emacs**
  - Alt 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.
Alt is interpreted as Alt on Microsoft Windows and can be used for shortcuts.

The \texttt{:meta} modifier is not used in an \texttt{output-pane input-model gesture specification}.

\texttt{Control} keystrokes are interpreted as accelerators. Standard accelerators are added for standard menu commands, for example \texttt{Ctrl+S for File > Save}.

The selection is deleted on input, and movement keys behave like a typical Microsoft Windows or KDE/Gnome editor.

Emacs \texttt{Control} keys are available, since they do not clash with the Macintosh \texttt{Command} key.

The selection is deleted on input, and movement keys behave like a typical Mac OS X editor.

By default \texttt{keys-style} is \texttt{:pc} on Microsoft Windows platforms and \texttt{:emacs} on Unix/Linux and Mac OS X platforms. You can supply methods for \texttt{interface-keys-style} on your own interface classes that override the default methods.

In the Cocoa implementation, \texttt{Command} keystrokes such as \texttt{Command+X} are available if there is a suitable \texttt{Edit} menu, regardless of the Editor emulation.

See the chapter "Emulation" in the \textit{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 \texttt{mod1} (see \texttt{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 `menu-bar-items` to its menus.

**Package**

`capi`

**Signature**

`interface-menu-groups interface => result`

**Arguments**

`interface` A CAPI interface.

**Values**

`result` A list.

**Description**

The generic function `interface-menu-groups` is called when an embedded document sets the menu bar of its containing interface.

Then, the menu bar for the embedded document includes three groups of menus that are supplied by the container (file-group, view-group, windows-group). `interface-menu-groups` is used to define these groups of menus.

`interface-menu-groups` 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.

The default method, on `interface`, simply returns `(nil nil nil)`.

**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`
interface-preserve-state  

**Generic Function**

**Summary**
Called before an interface is destroyed during session saving.

**Signature**
interface-preserve-state interface

**Arguments**
interface  
An interface.

**Description**
The generic function `interface-preserve-state` is called by `hcl:save-current-session` 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 `hcl:save-current-session` already started to destroy interfaces.

The return value is not used.

The default method does nothing.

**See also**
interface-preserving-state-p

---

interface-preserving-state-p  

**Function**

**Summary**
The predicate for whether an interface is in "preserving-state" context.

**Signature**
interface-preserving-state-p interface => result

**Arguments**
interface  
An interface.

**Values**
result  
t, :different-invocation or :keeping-processes.

**Description**
An interface enters "preserving-state" context just before it is destroyed by `hcl:save-current-session`, and exits the context just after `interface-display` 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 capi

Signature interface-reuse-p interface &rest initargs &key &allow-other-keys => reusep

Arguments

- interface: An instance of a subclass of interface.
- initargs: Initargs for interface.

Values reusep: A boolean

Description The generic function \texttt{interface-reuse-p} returns a true value if \texttt{interface} is suitable for reuse with \texttt{initargs}.

\texttt{interface-reuse-p} is used by \texttt{locate-interface} if no matching interface is found first by \texttt{interface-match-p}. In this case, when there is an interface for which \texttt{interface-reuse-p} returns true, then \texttt{locate-interface} reinitializes it by \texttt{reinitialize-interface} and returns it.

Notes \texttt{interface-reuse-p} should not be confused with \texttt{reuse-interfaces-p}, which determines the global re-use state.

See also \texttt{interface-match-p} \texttt{locate-interface}

\textbf{interface-toolbar-state}

\textbf{Function}

Signature interface-toolbar-state interface key => value 
(setf interface-toolbar-state) value interface key => value

Arguments

- interface: An instance of interface or a subclass.
- key: One of the toolbar-states plist keys.
- value: The value associated with the toolbar-states plist key.
Values

value

The value associated with the toolbar-states plist key.

Description

The functions `interface-toolbar-state` and `(setf interface-toolbar-state)` 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.

key can be one of the following, with the corresponding value:

:visible visible is true if the toolbar is visible and false if it is hidden. The default is true.

:items items is a list of the names of the toolbar-items which are shown on the toolbar, in the order they are shown. The built-in names :separator, :space and :flexible-space represent various kinds of gap between items. On Microsoft Windows, an item can be a list of the form (:titled-separator title) which starts a dockable group of items that displays title when it is undocked. The default items includes all items in toolbar-items, with :separator between each toolbar-component.

:display display is a keyword describing what is displayed for each item. It can be :image (just shows an image), :title (just shows the title), :image-and-title (shows both title and image) or :image-and-title-horizontal (shows title and image horizontally, only supported on GTK+). The default is platform-specific.
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 \textit{pane} is not a CAPI pane (that is, it is not an instance of a subclass of \texttt{element}, \texttt{collection} or \texttt{pinboard-object}).

Otherwise \texttt{interface-visible-p} returns \texttt{t}.

\textbf{Note:} On Microsoft Windows, \texttt{interface-visible-p} may return \texttt{t} even though the interface is entirely obscured by another window.

\textbf{interpret-description} \hfill \textit{Generic Function}

\begin{description}
\item[Summary] Converts an abstract description of a layout’s children into a list of the children’s geometry objects.
\item[Package] capi
\item[Signature] \texttt{interpret-description \ layout \ description \ interface}
\item[Description] The generic function \texttt{interpret-description} translates an abstract description of the \texttt{layout}’s children into a list of those children’s geometry objects.

For example, \texttt{column-layout} 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 \texttt{parse-layout-descriptor} to do the translation for each item.

\item[Example] See the examples in the directory \texttt{examples/capi/layouts/}.
\item[See also] \texttt{parse-layout-descriptor}
\texttt{define-layout}
\texttt{layout}
\texttt{interface}
\end{description}
invalidate-pane-constraints

**Function**

**Summary**
Causes the resizing of a pane if its minimum and maximum size constraints have changed. It returns t if resizing was necessary.

**Package**
capi

**Signature**
invalidate-pane-constraints pane

**Description**
This function informs the CAPI that pane’s constraints (its minimum and maximum size) may have changed. The CAPI then checks this, and if the pane is no longer within its constraints it resizes it so that it is and then makes the pane’s parent layout lay its children out and display them again at their new positions and sizes. If the pane is resized, then invalidate-pane-constraints returns t.

**See also**
get-constraints
layout
element
define-layout

invoke-command

**Function**

**Summary**
Invokes a command in the input model for a specified output pane.

**Package**
capi

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

**Description**
This invokes the command in the input model for the given output-pane, with the translator being called to process the gesture information. To avoid the translation, use invoke-untranslated-command.
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**
```lisp
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

- menu-item
- button
- item-pinboard-object
- popup-menu-button
- toolbar-button

Initargs

: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 text is nil, this is called to print the data.
:selected  If t the item is selected.

Accessors

- item-collection
- item-data
- item-text
- item-print-function
- item-selected

Description

An item can provide its own callbacks to override those specified in its enclosing collection, and can also provide some data to get passed to those callbacks.

An item is printed in the collection by print-collection-item. By default this returns a string using item's text if specified, or else calls a print function on the item's data. The print-function will either be the one specified in the item, or else the print-function for its parent collection.

The selected slot in an item is non-nil if the item is currently selected. The accessor item-selected is provided to access and to set this value.

Example

```
(defun main-callback (data interface)
  (capi:display-message "Main callback: ~S" data))

(defun item-callback (data interface)
  (capi:display-message "Item callback: ~S" data))
```
(capi:contain (make-instance 'capi:list-panel :items (list (make-instance 'capi:item :text "Item" :data '(some data) :selection-callback 'item-callback) "Non-Item 1" "Non-Item 2") :selection-callback 'main-callback))

See also itemp collection choice print-collection-item

itemp

Generic Function

Package capi

Signature itemp object

Description This is equivalent to

(typep object 'capi:item)

See also item collection

item-pinboard-object

Class

Summary An item-pinboard-object is a pinboard-object that displays a single piece of text.

Package capi
Superclasses  

item

Description

The item-pinboard-object displays an item on a pinboard layout. It displays the text specified by the item in the usual way (either by the text field, or through printing the data with the print function).

Example

(capi:contain (make-instance 'capi:item-pinboard-object :text "Hello World"))

(capi:contain (make-instance 'capi:item-pinboard-object :data :red :print-function 'string-capitalize))

See also

image-pinboard-object
pinboard-layout

labelled-arrow-pinboard-object

Class

Package  
capi

Superclasses  
arrow-pinboard-object
labelled-line-pinboard-object

Description

A subclass of pinboard-object which displays an arrow and draws a label on it.

Example

See labelled-line-pinboard-object.

See also

pinboard-layout

labelled-line-pinboard-object

Class

Summary

A subclass of pinboard-object which draws a labelled line.
Package: capi

Superclasses:
- item-pinboard-object
- line-pinboard-object

Subclasses:
- labelled-arrow-pinboard-object

Initargs:
- :text-foreground
  The color of the label text.

Accessors:
- labelled-line-text-foreground

Description:
A subclass of pinboard-object which displays a line and draws a label in the middle of it.
Note that the label text is inherited from item.

Example:
```lisp
(capi:contain
 (make-instance 'capi:pinboard-layout
 :description (list (make-instance 'capi:labelled-line-pinboard-object
 :text "Labelled Line"
 :start-x 10 :start-y 10
 :end-x 80 :end-y 60)
 (make-instance 'capi:labelled-arrow-pinboard-object
 :text "Labelled Arrow"
 :start-x 10 :start-y 70
 :end-x 80 :end-y 120
 :head-direction :both))))
```

See also:
- pinboard-layout

layout

Class

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

Superclasses:
- titled-object
- simple-pane

Subclasses:
- simple-layout
- grid-layout
- pinboard-layout
- switchable-layout

Initargs:
- :default
  A flag to mark the default layout for an interface.
- :description
  The list of the layout’s children.
- :initial-focus
  A child of the layout, or its name, specifying where the input focus should be, or nil.

Accessors:
- layout-description

Description:
The layout’s description is an abstract description of the children of the layout, and each layout defines its format.

Generally, description is a list, each element of which is one of:

- a pane
- a slot name, where the name refers to a slot in the layout’s interface containing a pane
- a string, where the string gets converted to a title-pane

For grid-layout and its subclasses, elements of description can also be nil. See grid-layout for the interpretation of this value.

Setting the layout description causes the layout to translate it, and then to layout the new children, adjusting the size of its parent if necessary.

A number of default layouts are provided which provide the majority of layout functionality that is needed. They are as follows:
**simple-layout**  A layout for one child.

**row-layout**  Lays its children out in a row.

**column-layout**  Lays its children out in a column.

**grid-layout**  Lays its children out in an n by m grid.

**pinboard-layout**  Places its children where the user specifies.

**switchable-layout**  Keeps only one of its children visible.

*initial-focus* specifies which child of the layout has the input focus when the layout is first displayed. Panes are compared by eq or 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,

```lisp
(setq (capi:layout-description pinboard-layout)
  (cons object
    (capi:layout-description pinboard-layout))))
```

is equivalent to

```lisp
(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  

<table>
<thead>
<tr>
<th>Superclasses</th>
</tr>
</thead>
<tbody>
<tr>
<td>pinboard-object</td>
</tr>
</tbody>
</table>

Subclasses  

<table>
<thead>
<tr>
<th>Subclasses</th>
</tr>
</thead>
<tbody>
<tr>
<td>arrow-pinboard-object</td>
</tr>
<tr>
<td>right-angle-line-pinboard-object</td>
</tr>
</tbody>
</table>

Initargs  

<table>
<thead>
<tr>
<th>Initargs</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>:start-x</td>
<td>The x coordinate of the start of the line.</td>
</tr>
<tr>
<td>:start-y</td>
<td>The y coordinate of the start of the line.</td>
</tr>
<tr>
<td>:end-x</td>
<td>The x coordinate of the end of the line.</td>
</tr>
<tr>
<td>:end-y</td>
<td>The y coordinate of the end of the line.</td>
</tr>
</tbody>
</table>

Description  

start-x, start-y, end-x and end-y default to values computed from the x, y, width and height. They are used to compute the size of the object, and the proper value of x and y. Note that width and height may be larger, for example to accommodate the label in a labelled-line-pinboard-object, and the x and y are adjusted for that.

To change the end points of the line, call move-line.

A complementary class right-angle-line-pinboard-object is provided which draws a line around the edge of the pinboard object.

Example  

```lisp
(capi:contain
 (make-instance 'capi:line-pinboard-object
 :start-x 0 :end-x 100
 :start-y 100 :end-y 0))
```

See also  

move-line
pinboard-layout

---

**line-pinboard-object-coordinates**  

Function

Summary  

Returns the coordinates of a line-pinboard-object.

Package  

capi
Signature

\texttt{line-pinboard-object-coordinates \textit{object} \rightarrow \textit{start-x}, \textit{start-y}, \textit{end-x}, \textit{end-y}}

Arguments

\textit{object} \quad \text{A line-pinboard-object.}

Values

\textit{start-x} \quad \text{An integer.}
\textit{start-y} \quad \text{An integer.}
\textit{end-x} \quad \text{An integer.}
\textit{end-y} \quad \text{An integer.}

Description

The function \texttt{line-pinboard-object-coordinates} returns the start and end coordinates of the \texttt{line-pinboard-object}.

See also

move-line

\textbf{list-panel}

\textit{Class}

Summary

The class \texttt{list-panel} is a pane that can display a group of items and provides support for selecting items and performing actions on them.

Package

capi

Superclasses

\texttt{choice}  \\
\texttt{simple-pane}  \\
\texttt{sorted-object}  \\
\texttt{titled-object}

Subclasses

\texttt{list-view}  \\
\texttt{multi-column-list-panel}

Initargs

:\texttt{right-click-selection-behavior}

A keyword or \texttt{nil}. 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 \texttt{filter} is non-nil.

:filter-automatic-p
   A boolean. The default value is \texttt{t}.

:filter-callback
   A function designator or the keyword \texttt{: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, \texttt{t} or \texttt{nil}.

:filter-help-string
   A string, \texttt{t} or \texttt{nil}.

Accessors

\texttt{list-panel-right-click-selection-behavior}

Description

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

The \texttt{list-panel} class does not support the \texttt{:no-selection} interaction style. For a non-interactive list use a \texttt{display-pane}.

To scroll a \texttt{list-panel}, call \texttt{scroll} with \texttt{scroll-operation :move}. \texttt{mnemonic-title} is interpreted as for \texttt{menu}.

\texttt{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

\[
\text{color-function list-panel item state} \Rightarrow \text{result}
\]

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**

```lisp
(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

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

Package capi
Signature  
list-panel-enabled list-panel => enabledp

Signature  
(setf list-panel-enabled) enabledp list-panel => enabledp

Arguments  
list-panel A list-panel.

Values  
enabledp A boolean.

Description  
The generic function list-panel-enabled determines whether list-panel is currently enabled. It is equivalent to the simple-pane accessor simple-pane-enabled.

The generic function (setf list-panel-enabled) enables list-panel when enabledp is true, and disables it otherwise. It is equivalent to (setf simple-pane-enabled).

See also  
simple-pane

---

list-panel-filter-state  
Generic Function

Summary  
Accesses the state of the filter in a filtered list-panel.

Signature  
list-panel-filter-state list-panel => filter-state

(setf list-panel-filter-state) new-state list-panel

Description  
The generic function list-panel-filter-state accesses the state of the filter in a filtered list-panel (that is, a list-panel created with filter t).

list-panel-filter-state returns the state of the filter in list-panel. The return value filter-state is the same type as the state that is used in filtering-layout.

(setf list-panel-filter-state) sets the filter in list-panel, filters the unfiltered items and displays those that match the new-state. The new-state has the same semantics as the new-value of (setf filtering-layout-state). It can be a result
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

**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)
```

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

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 :all.

:use-large-images
   Indicates whether large icons will be used (generally only if the icon view will be used). Defaults to t.

:use-small-images
   Indicates whether small icons will be used. Defaults to t.

:use-state-images
   Indicates whether state images will be used. Defaults to nil.

:large-image-width
   Width of a large image. Defaults to 32.

:large-image-height
   Height of a large image. Defaults to 32.

:small-image-width
   Width of a small image. Defaults to 16.

:small-image-height
   Height of a small image. Defaults to 16.

:state-image-width
   Width of a state image. Defaults to small-image-width.

:state-image-height
   Height of a state image. Defaults to small-image-height.
Accessors

list-view-view
list-view-subitem-function
list-view-subitem-print-functions
list-view-image-function
list-view-state-image-function
list-view-columns
list-view-auto-reset-column-widths

Description

The list view inherits its functionality from choice. In many ways it may be regarded as a kind of enhanced list panel, although its behavior is not identical. It supports single selection and extended selection interactions.

The list view displays its items in one of four ways, determined by the value in the view slot. An application may use the list view pane in just a single view, or may change the view between all four available views using (setf list-view-view).

See the notes below on using both large and small icon views.

In all views, the text associated with the item (the label) is returned by the print-function, as with any other choice.

- The icon view — :icon
  In this view, large icons are displayed, together with their label, positioned in the space available.

- The small icon view — :small-icon
  In this view, small icons are displayed, together with their label, positioned in the space available.

- The list view — :list
  In this view, small icons are displayed, arranged in vertical columns.
• The report view — :report

In this view, multiple columns are displayed. A small icon and the item’s label is displayed in the first column. Additional pieces of information, known as subitems, are displayed in subsequent columns.

To use the view :report, columns must specify a list of column specifiers. Each column specifier is a plist, in which the following keywords are valid:

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>:title</td>
<td>The column heading</td>
</tr>
<tr>
<td>:width</td>
<td>The width of the column in pixels. If this keyword is omitted or has the value nil, the width of the column is automatically calculated, based on the widest item to be displayed in that column.</td>
</tr>
<tr>
<td>:align</td>
<td>May be :left, :right or :center to indicate how items should be aligned in this column. The default is :left. Only left alignment is available for the first column.</td>
</tr>
</tbody>
</table>

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.
:small
Specifies an image-list object that contains the small 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 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 | interactive-pane
---|---
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 *standard-output* is sent to the listener, and finally the results of the evaluation are displayed.
Example | (capi:contain (make-instance 'capi:listener-pane) :best-width 300 :best-height 200)
See also | collector-pane, interactive-pane

### listener-pane-insert-value

**Function**

**Summary** | Evaluates a form and inserts the result in a listener-pane.

**Package** | capi

**Signature** | listener-pane-insert-value pane form

**Arguments**

- pane | A listener-pane.
- form | A Lisp form.

**Description** | The function listener-pane-insert-value evaluates the form form and inserts the result in the listener-pane pane, as if it resulted from user input. The result is printed, and the values of the history variables *, **, ***, /, //, and /// are set.

listener-pane-insert-value may be called in any process. Multiple values in the result of evaluating form are not supported: the first value only is inserted in pane.

**See also** | interactive-pane-execute-command
### load-cursor

**Function**

**Summary**
Loads a cursor.

**Package**
capi

**Signature**
load-cursor filename-or-list => cursor

**Arguments**
filename-or-list  A string or a list.

**Values**
cursor  A cursor object.

**Description**
The function `load-cursor` loads a cursor from your cursor file, or loads a built-in cursor. It returns a cursor object which can be supplied as the value of the `simple-pane :cursor` initarg.

The cursor object can also be set with `(setf simple-pane-cursor)` to change a pane’s cursor. This must be done in the process of the pane’s interface.

If `filename-or-list` is a string, then it names a file which should be in a suitable format for the platform, 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 cur1 (capi:load-cursor "arrow_l"))
```

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.
The loaded sound sound will be unloaded (freed) automatically when its owner is destroyed. To create a sound that is never unloaded, pass the screen as the argument owner.

See also free-sound
play-sound
read-sound-file

**locate-interface**

**Generic Function**

**Summary**
Finds an interface of a given class that matches supplied initargs.

**Package**
capi

**Signature**
```
locate-interface class-spec &rest initargs
  &key screen no-busy-interface
  &allow-other-keys => interface
```

**Arguments**
- `class-spec` A specifier for a subclass of interface.
- `initargs` Initialization arguments for class-spec.
- `screen` A screen or nil.
- `no-busy-interface` A boolean, defaulting to nil.

**Values**
- `interface` An interface of class class-spec, or nil.

**Description**
The generic function `locate-interface` finds an interface of the class specified by `class-spec` that matches `initargs` and `screen`.

First, `locate-interface` finds all interfaces of the class specified by `class-spec` by calling `collect-interfaces` with `class-spec` and `screen`. The first of these which match `initargs` (by `interface-match-p`) is returned.
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

**Summary**
The `lower-interface` function pushes the window containing a specified pane to the back of the screen.

**Package**
capi

**Signature**
`lower-interface pane`

**Description**
This pushes the window containing `pane` to the back of the screen. To bring it back use `raise-interface`, and to iconify it use `hide-interface`.

See also `hide-interface`  
`interface`  
`lower-interface`  
`raise-interface`  
`quit-interface`
make-container

Generic Function

Summary
The generic function make-container creates a container for a specified element.

Package
capi

Signature
make-container element &rest interface-args

Description
This creates a container for element such that calling display on it will produce a window containing element on the screen. It will produce a container for any of the following classes of object:

  simple-pane
  layout
  interface
  pinboard-object
  menu
  menu-item
  menu-component
  list

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

The arguments interface-args will be passed through to the make-instance of the top-level interface, assuming that pane is not a top-level interface itself.

The complementary function contain uses make-container to create a container for an element which it then displays.

Example
(capi:display (capi:make-container
  (make-instance
    'capi:text-input-pane)))
See also    contain
display
interface
element

\textbf{make-docking-layout-controller} \hspace{1cm} \textit{Function}

\begin{description}
\item[Package] capi
\item[Signature] \texttt{make-docking-layout-controller} \texttt{=} \texttt{controller}
\item[Values] \texttt{controller} \hspace{1cm} \text{A docking layout controller.}
\item[Description] The function \texttt{make-docking-layout-controller} returns a docking layout controller object for use as the \texttt{controller} ini-
targ in \texttt{docking-layout}.
Layouts which share a docking layout controller are known as a Docking Group. See \texttt{docking-layout} for information about Docking Groups.
\end{description}

See also \texttt{docking-layout}

\textbf{make-foreign-owned-interface} \hspace{1cm} \textit{Function}

\begin{description}
\item[Summary] Creates a dummy interface which allows another application’s window to be the owner of a CAPI dialog.
\item[Package] capi
\item[Signature] \texttt{make-foreign-owned-interface} \texttt{&key handle name} \texttt{=} \texttt{interface}
\item[Arguments] \texttt{handle} \hspace{1cm} \text{A Microsoft Windows hwnd.}
\texttt{name} \hspace{1cm} \text{A string naming \texttt{interface}.}
\end{description}
Values

interface  An instance of foreign-owned-interface.

Description

The function make-foreign-owned-interface creates an instance of foreign-owned-interface. interface can be used as the owner argument when displaying a dialog. For information about dialog owners, see the "Prompting for Input" chapter in the LispWorks CAPI User Guide.

handle must be supplied and is the window handle (Windows hwnd) of a window in some application. For a CAPI window this window handle can be obtained by simple-pane-handle. 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.

name becomes the name of interface, and has no other meaning.

make-foreign-owned-interface is implemented only on Microsoft Windows.

Example

This example shows how a CAPI window can be the owner of a dialog in another LispWorks image.

Start LispWorks for Windows.

1. In the Listener, do Tools > Interface > Listen. This puts the Listener interface in the value of *.

2. In the Listener enter (capi:simple-pane-handle *). The returned value is the window handle, it should be an integer. Denote this value by hwnd.

Start another LispWorks for Windows image (do not quit the first image). In the Listener of this second LispWorks image:

1. Enter (setq foi (capi:make-foreign-owned-interface :handle hwnd)).

2. Enter (capi:prompt-for-color "Color?" :owner foi).
Now note that the Color dialog is owned by the Listener of the first LispWorks image.

### make-general-image-set

**Function**

**Summary**

Creates an `image-set` object.

**Package**

`capi`

**Signature**

```
make-general-image-set &key image-count width height id => image-set
```

**Arguments**

- `image-count` An integer.
- `width` An integer or `nil`.
- `height` An integer or `nil`.
- `id` A pathname, string or symbol.

**Values**

`image-set` An `image-set` object.

**Description**

The `make-general-image-set` function creates an `image-set` object that refers to an image or a file containing an image.

`id` is a pathname or string identifying an image file, or a symbol previously registered with `register-image-translation`.

`width` and `height` are the dimensions of a single sub-image within the main image, and `image-count` specifies the number of sub-images in the image.

**Example**

See the files

- `examples/capi/choice/tree-view.lisp`
- `examples/capi/choice/extended-selection-tree-view.lisp`
- `examples/capi/elements/toolbar.lisp`
See also  
image-set  
make-resource-image-set

make-icon-resource-image-set  
Function

Summary  
Constructs an image set object identifying a icon resource in a DLL.

Package  
capi

Signature  
make-icon-resource-image-set &key image-count width height  
library id => image-set

Arguments  
image-count  
An integer.
width  
An integer.
height  
An integer.
library  
A string.
id  
A string or an integer.

Values  
image-set  
An image-set object.

Description  
The make-icon-resource-image-set function is only available in LispWorks for Windows. It constructs an image set object that identifies an image stored as a icon resource in a DLL.

width and height are the dimensions of a single sub-image within the main image, and image-count specifies the number of sub-images in the image.

library should be a string specifying the name of the DLL.
id should be either an integer which is the resource identifier of the icon, or a string naming the icon resource.
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.
component-p A boolean.

Values menu A menu or a menu-component.

Description The function make-menu-for-pane makes a menu or a menu-component for the pane pane with the items specified by items. items should be a list in which each element is a menu-item, menu-component or menu. title and menu-name provide a title and name for menu. title and menu-name both default to nil.

If component-p is true, then make-menu-for-pane creates a menu-component rather than a menu. The default value of component-p is nil.

menu is set up so that by default each callback inside it is done on the pane pane itself. This is the useful feature of make-menu-for-pane because it avoids the need to set up items to do their callbacks on pane explicitly.

Note that this is merely the default behavior. You can specify different callback behavior on a per-item basis, using setup-callback-argument and callback-data-function (see menu-object), callback-type (see callbacks) and data for menu-item (see item).

See also make-pane-popup-menu
    pane-popup-menu-items

make-pane-popup-menu Generic Function

Summary Generates a popup menu or menu-component.

Package capi

Signature make-pane-popup-menu pane interface
    &key title menu-name component-p => menu
Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pane</td>
<td>A pane in an interface.</td>
</tr>
<tr>
<td>interface</td>
<td>An interface or nil.</td>
</tr>
<tr>
<td>title</td>
<td>A string or nil.</td>
</tr>
<tr>
<td>menu-name</td>
<td>A string or nil.</td>
</tr>
<tr>
<td>component-p</td>
<td>A boolean.</td>
</tr>
</tbody>
</table>

Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>menu</td>
<td>A menu or a menu-component.</td>
</tr>
</tbody>
</table>

Description

The generic function `make-pane-popup-menu` generates a popup menu for `pane`.

`interface` can be `nil` if `pane` has already been created, in which case the `interface` of `pane` is used (obtained by the `element` accessor `element-interface`).

`title` and `menu-name` provide a title and name for `menu`. `title` and `menu-name` both default to `nil`.

If `component-p` is true, then `make-pane-popup-menu` creates a `menu-component` rather than a `menu`. The default value of `component-p` is `nil`.

Example

This code makes an interface with two `graph-panes`. The `initialize-instance` method uses `make-pane-popup-menu` to add a menu to the menu bar from which the user can perform operations on the graphs.

Note that, because `make-pane-popup-menu` calls `make-menu-for-pane` 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))

(defun 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
make-resource-image-set

**Signature**

```lisp
(make-resource-image-set &key image-count width height
                        library id => image-set)
```

**Arguments**

- `image-count`: An integer.
- `width`: An integer.
- `height`: An integer.
- `library`: A string.
- `id`: A string or an integer.

**Values**

- `image-set`: An `image-set` object.

**Description**

The `make-resource-image-set` 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.

`width` and `height` are the dimensions of a single sub-image within the main image, and `image-count` specifies the number of sub-images in the image.

`library` should be a string specifying the name of the DLL.

`id` should be either an integer which is the resource identifier of the bitmap, or a string naming the bitmap resource.

**See also**

- `image-set`
- `make-icon-resource-image-set`
- `make-general-image-set`

---

**make-scaled-general-image-set**

**Function**

**Summary**

Constructs an image set object which scales images in another image set.

**Package**

capi

**Signature**

```lisp
(make-scaled-general-image-set &key width
                               height id => image-set)
```
Arguments

- **width**  
  An integer.

- **height**  
  An integer.

- **id**  
  A pathname, string or symbol.

Values

- **image-set**  
  An **image-set** object.

Description

The `make-scaled-general-image-set` function is only available in LispWorks for Windows. It constructs an image set that provides scaled images based on an **image-set** object constructed from `id` as if by `make-general-image-set`. `width` and `height` are the dimensions of a single sub-image within the main image. That is, the sub-images are scaled to this size.

See also

- **image-set**

- **make-general-image-set**

**make-scaled-image-set**

Function

Summary

Creates an image set by scaling the images of another image set.

Package

`capi`

Signature

```
make-scaled-image-set &key image-count width  
  height base-image-set => image-set
```

Arguments

- **image-count**  
  An integer.

- **width**  
  An integer.

- **height**  
  An integer.

- **base-image-set**  
  An image set.

Values

- **image-set**  
  An **image-set** object.
The `make-scaled-image-set` function is only available in LispWorks for Windows. It constructs an image set that provides scaled images based on an existing image set object `base-image-set`.

`width` and `height` are the dimensions of a single sub-image within the main image. That is, the sub-images in `base-image-set` are scaled to this size to produce the sub-images of `image-set`.

`image-count` specifies the number of sub-images in the image.

See also

- `image-set`
- `make-general-image-set`

### make-sorting-description

- **Function**

- **Summary**: Makes a sort description suitable for use in a `sorted-object`.

- **Package**: capi

- **Signature**: `make-sorting-description &key type key sort reverse-sort sort-function => sorting-description`

- **Arguments**
  - `type` A Lisp object naming the type of sorting.
  - `key` A function of one argument.
  - `sort` A function of two arguments.
  - `reverse-sort` A function of two arguments.
  - `sort-function` A sorting function.

- **Description**: The function `make-sorting-description` makes a sort description object that can be used as one of the `sort-descriptions` in a `sorted-object` such as a `list-panel`.

  `type` is a name that should be unique amongst the `sort-descriptions` of a `sorted-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

\[
\text{sort-function \ items \ predicate \ &key \ key}
\]

The default value of *sort-function* is **sort**.

**Example**

```lisp
(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.


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

```lisp
(setq pl
  (capi:contain
    (make-instance 'capi:pinboard-layout
      :visible-min-height 500
      :visible-min-width 200))))
```

Add some pinboard-objects:
(capi:apply-in-pane-process
pl #'(lambda (pp)
  (dotimes (y 10)
    (let ((yy (* y 40)))
      (capi:manipulate-pinboard
       pp
       (make-instance 'capi:line-pinboard-object
                      :start-x 4 :start-y yy
                      :end-x 54 :end-y (+ 6 yy))
       :add-top)
      (capi:manipulate-pinboard
       pp
       (make-instance 'capi:pinboard-object
                      :x 4 :y (+ 20 yy)
                      :width 50 :height 6
                      :graphics-args
                      '(:background :red))
       :add-top))))
pl)

Remove some pinboard-objects:

(capi:apply-in-pane-process
pl #'(lambda (pp)
  (dotimes (y 15)
    (let ((po (capi:pinboard-object-at-position pp 10
          (* y 30))))
      (when po (capi:manipulate-pinboard pp po
                      :delete)))))))
pl)

Remove all line-pinboard-objects:

(capi:apply-in-pane-process
pl 'capi:manipulate-pinboard pl
#'(lambda (x)
    (typep x 'capi:line-pinboard-object))
    :delete-if)

See also

pinboard-layout
set-object-automatic-resize
map-collection-items

Generic Function

Summary
The generic function map-collection-items calls a specified function on all the items in a collection.

Package
capi

Signature
map-collection-items collection function &optional collect-results-p

Arguments
collection A collection.
function A function designator for a function of one argument.
collect-results-p A generalized boolean.

Description
Calls function on each item in the collection by calling the collection’s items-map-function. If collect-results-p is true, the results of these calls are returned in a list.

Example
(setq collection (make-instance 'capi:collection :items '(1 2 3 4 5)))
(capi:map-collection-items collection 'princ-to-string t)

See also
collection
choice

map-pane-children

Generic Function

Summary
Calls a function on each of a pane’s children.

Package
capi

Signature
map-pane-children pane function &key visible test reverse

Arguments
pane A CAPI pane.
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.

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 ()
       (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

Summary
Calls a function on each of the descendant panes of a pane.

Package
capi

Signature
map-pane-descendant-children pane function &key visible test reverse leaf-only

Arguments
\textit{pane} \quad \text{A CAPI pane.}

\textit{function} \quad \text{A function of one argument.}

\textit{visible} \quad \text{A boolean. The default value is nil.}

\textit{test} \quad \text{A function of one argument, or nil. The default is nil.}

\textit{reverse} \quad \text{A boolean. The default value is nil.}

\textit{leaf-only} \quad \text{A generalized boolean. The default value is nil.}

Description
The function \texttt{map-pane-descendant-children} applies \texttt{function} to \texttt{pane}'s descendant panes (that is, the children and each of their children recursively), depth first.

If \textit{visible} is true, then \texttt{function} is applied only to the visible descendant panes.

If \textit{test} is non-nil, it is a function which is applied first to each descendant pane, and only those for which \textit{test} returns a true value are then passed to \texttt{function}.

If \textit{reverse} is non-nil, the order in which the children are processed is reversed.

If \textit{leaf-only} is true, then \texttt{function} is applied only to those panes which do not have children.

See also
\texttt{map-pane-children}
\texttt{pane-descendant-child-with-focus}
map-typeout Function

Package capi

Signature map-typeout pane &rest args

Description Makes a collector-pane the visible child of a switchable-layout, and returns it as well. The switchable layout is found by looking up the parent hierarchy starting from pane.

The switchable layout should have one or more children. If it has one child, a new collector pane is made using args as the initargs with buffer-name defaulting to "Background Output". If it has more than one, it searches through the children to find the first collector pane.

See also unmap-typeout
with-random-typeout
collector-pane

*maximum-moving-objects-to-track-edges* Variable

Summary Limits the tracking of edges in a graph.

Package capi

Initial Value 15

Description If there are more than *maximum-moving-objects-to-track-edges* objects being moved in a graph, then edges are not tracked.

The value should be an integer.
menu

Class

Summary
The class menu 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
capi

Superclasses
element
titled-menu-object

Initargs
:items
The items to appear in the menu.

:items-function
A function to dynamically compute the items.

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

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

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

:image-function
A function providing images for the menu items, or nil.

Accessors
menu-items
menu-image-function

Description
A menu has a title, and has items appearing in it, where an item can be either a menu-item, a menu-component or another menu.
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.
- `nil`: A character is chosen from a list of common mnemonics, or the :default behavior is followed. This is the default.
- :default: A mnemonic is chosen using some rules.
- :none: 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 \textit{mnemonic-title}. To make \textit{mnemonic-escape} itself appear in the button, precede it in \textit{mnemonic-title} with \textit{mnemonic-escape}. For example:

\texttt{\textbackslash :mnemonic-title "&Compile &\& Load File..."}

If \textit{image-function} is non-nil, it should be a function of one argument. \textit{image-function} is called with the data of each menu item and should return one of:

\begin{itemize}
  \item \texttt{nil} \hspace{1cm} No image is shown.
  \item An \texttt{image} object \hspace{1cm} The menu displays this image.
  \item An image id or \texttt{external-image} \hspace{1cm} The system converts the value to a temporary \texttt{image} for the menu item and frees it when it is no longer needed.
\end{itemize}

If \textit{image-function} is \texttt{nil}, no items in the menu have images. This is the default value.

\textbf{Notes}

1. On Cocoa and GTK+, menu items can contain both images and strings, so the \textit{print-function} should return the appropriate string or \texttt{""} 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 \texttt{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 \texttt{contain} will do just that for you.

3. To display a menu as a context (right button) menu, use \texttt{display-popup-menu}, and to display a menu via a labelled button use \texttt{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

```lisp
(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:

```lisp
(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`:

```lisp
(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:

```lisp
(capi:contain (make-instance 'capi:menu :title "Mnemonic Title" :mnemonic 1 :items '(1 2 3)))
(capi:contain (make-instance 'capi:menu :mnemonic-title "Mnemonic 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  
capi

Superclasses  
choice  
titled-menu-object

Initargs  
:items  The items to appear in the menu.
:items-function  A setup callback function to dynamically compute the items.
:selection-function  A setup callback function to dynamically compute the selection.
:selected-item-function  A setup callback function to dynamically compute the selected item.
:selected-items-function  A setup callback function to dynamically compute the selected items.

Description  
Because menu-component is a choice, the component can have interaction :no-selection, :single-selection or :multiple-selection (extended selection does not apply here). This is represented visually in the menu as appropriate to the window system that the CAPI is running on (by ticks in Microsoft Windows, and by radio buttons and check buttons in Motif).

Note that it is not appropriate to have menu components or submenus inside :single-selection and :multiple-selection components, but it is OK in :no-selection components.

items and items-function behave as in menu.
No more than one of selection-function, selected-item-function and selected-items-function should be non-nil. Each defaults to nil. If one of these setup callbacks is supplied, it should be a function which is called before the menu-component is displayed and which determines which items are selected. By default the setup callback is called on the interface of the menu-component, but this argument can be changed by passing the menu-object initarg setup-callback-argument.

selection-function, if non-nil, should return a list of indices suitable for passing to the choice accessor (setf choice-selection).

selected-item-function, if non-nil, should return an object which is an item in the menu-component, or is equal to such an item when compared by the menu-component's test-function.

selected-items-function, if non-nil, should return a list of such objects.

Example

```lisp
(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.
Readers

help-key

Description

The text displayed in the menu item is the contents of the text slot, or the contents of the title slot, otherwise it is the result of applying the print-function to the data.

If selected-function is non-nil it should a function which is called before the menu-item is displayed and which determines whether or not the menu-item is selected. By default selected-function is called on the interface of the menu-item, but this argument can be changed by passing the menu-object initarg setup-callback-argument. The default value of selected-function is nil.

Callbacks are made in response to a user gesture on a menu-item. The callback-type (see callbacks), callback and callback-data-function (see menu-object) are found by looking for a non-nil value, first in the menu-item, then the menu-component (if any) and finally the menu. This allows a whole menu to have, for example, callback-type :data without having to specify this in each item. Some items could override this by having their callback-type slot non-nil if needed.

To specify a mnemonic in the menu item, you can use the initarg :mnemonic, or the initargs :mnemonic-title and :mnemonic-escape. These initargs are all interpreted just as in menu.

A menu item should not be used more in more than one place at a time.

help-key is interpreted as described for element.

accelerator can be a character or string specifying a key gesture which will be the accelerator for the menu item.
Note that both-case-p characters are not allowed with the single modifier Shift in the accelerator argument. So instead of

:accelerator "shift-x"

use

:accelerator "X"

Note that the Shift modifier still appears in the menu.

A both-case-p character is allowed with Shift if there are other modifiers, for example

:accelerator "alt-shift-x"

If accelerator is a character then the system adds the normal modifier for the platform. That is, Command on Cocoa and Control on Microsoft Windows. The shortcut is validated for the platform.

If accelerator is a string with modifier keys then the system uses it only if it follows the normal conventions for the platform. The shortcut is validated for the platform.

The special virtual modifier name "accelerator" is allowed in string values of accelerator. It is interpreted as the normal modifier key for the platform. For example:

:accelerator "accelerator-x"

means Control+X on Microsoft Windows and Motif, and Command+X on Cocoa.

If accelerator is a plist then its keys are keywords naming some or all of the supported libraries (as returned by default-library). The plist’s values are characters or strings which the system interprets as above, except that no check is made that the keyboard shortcut is valid for the platform.
accelerator has a special default value :default, which means that, depending on interface-keys-style for the interface, a standard accelerator is added if the item title matches a standard menu command.

alternative, when true, makes the menu-item an "alternative item". Alternative items are invoked if modifiers are held while selecting the "main item". These modifiers are defined by the item’s accelerator. The main item is the one before the first alternative item, and each alternative item must be within the same menu and menu component. For an example see examples/capi/elements/accelerators.lisp and for more information see the section "Alternative menu items" in the LispWorks CAPI User Guide.

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:

- t The item is enabled whenever there is a dialog.

- nil The item is disabled whenever there is a dialog.

- :same-as-normal Do the same as when there is no dialog. This depends on the enabled-function (see menu-object).

- A function A function that is called instead of the enabled-function 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 initarg :setup-call-back-argument (see menu-object 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**

```lisp
(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
The class menu-object is the superclass of all menu objects, and provides functionality for handling generic aspects of menus, menu components and menu items.

Package capi

Superclasses callbacks

Subclasses titled-menu-object

Initargs

:popup-callback

Callback before the menu appears.

:enabled-function

Returns true if the menu is enabled.

:enabled-slot

The object is enabled if the slot is non-nil.

:callback

The selection callback for the object.

:callback-data-function

A function to return data for the callback.

:setup-callback-argument

If non-nil, specifies the argument to the setup callbacks (listed below) that are used to set up the menu-object.

Accessors menu-popup-callback

Readers menu-object-enabled

Description When the menu object is about to appear on the screen, the CAPI does the following:
1. The setup callback \textit{items-function} (if there is one) is called and the result is used to set the items, for \texttt{menu} and \texttt{menu-component}. The argument passed to \textit{items-function} is the same as for the other setup callbacks (see below).

2. The \textit{popup-callback} (if there is one) is called and can make arbitrary changes to that object. The \textit{popup-callback} is always called with the menu object, regardless of the value of \texttt{setup-callback-argument}.

3. The other setup callbacks are called to set up the selection, enabled state and title. These setup callbacks include \texttt{enabled-function} for all \texttt{menu-objects} and \texttt{title-function} for all \texttt{titled-menu-objects}. The additional setup callbacks for \texttt{menu-component} are \texttt{selection-function}, \texttt{selected-item-function}, and \texttt{selected-items-function}. \texttt{menu-item} has the additional setup callback \texttt{selected-function}.

By default \texttt{setup-callback-argument} is \texttt{nil}, which means that each of the setup callbacks is called on the interface of the \texttt{menu-object}. If \texttt{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 \texttt{enabled-slot} is a short-hand means of creating an \texttt{enabled-function} which checks the value of a slot in the menu object’s interface.

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

The \texttt{callback} argument is placed in the \texttt{selection-callback}, \texttt{extend-callback} and \texttt{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

```lisp
(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

```lisp
merge-menu-bars frame document => menus
```

### Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>frame</code></td>
<td>A <code>document-frame</code>.</td>
</tr>
<tr>
<td><code>document</code></td>
<td>An <code>interface</code> or <code>nil</code>.</td>
</tr>
</tbody>
</table>

### Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>menus</code></td>
<td>A list of <code>menu</code> objects.</td>
</tr>
</tbody>
</table>

### Description

The generic function `merge-menu-bars` is called by the system to compute the menu bar for a `document-frame` interface.

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.

There is a built-in unspecialized method that appends the menu bars of the two interfaces and is equivalent to this:

```lisp
(defmethod capi:merge-menu-bars ((frame t) (document t))
    (append
    (capi:interface-menu-bar-items frame)
    (and document
        (capi:interface-menu-bar-items document)))))
```

You can customize the menu bar by adding methods which specialize on particular frame and document interface classes.

### See also

- `document-frame`
- `interface`
- `menu`

---

### message-pane

**Class**

**Summary**

The class displaying the message when a pane is created with the `:message` initarg.

**Package**

capi

---

307
Superclasses title-pane

Description The class message-pane is used to implement the message decoration on subclasses of titled-object.

A message-pane with text "Message" is created automatically when a titled-object is created with message "Message".

See also titled-object

modify-editor-pane-buffer Function

Summary The modify-editor-pane-buffer function allows you to modify the contents and fill mode of a specified buffer.

Package capi

Signature modify-editor-pane-buffer pane &key contents flag
fill fixed-fill force

Description The modify-editor-pane-buffer function modifies the editor-pane pane according to the keyword arguments.

The argument contents (if non-nil) supplies a new string to place in the buffer.

flag, if given, sets the flag slot of the editor buffer, which is used to mark it for various specialized uses.

If fill is non-nil the editor fills each paragraph in the buffer. If fill is a fixnum then the buffer is filled at that width. If fill is :default (the default value) and fixed-fill is supplied then the value fixed-fill is used. Otherwise the buffer is filled to the window width.

fixed-fill defaults to nil.

See also editor-pane
mono-screen

Class

Summary

The mono-screen class is created for monochrome screen.

Package
capi

Superclasses
screen

Description

This is a subclass of screen 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
color-screen

move-line

Generic Function

Summary

Moves a line-pinboard-object.

Package
capi

Signature
move-line line-pinboard-object start-x start-y end-x end-y &key redisplay

Arguments

line-pinboard-object

An instance of line-pinboard-object or a subclass.

start-x
The x coordinate of the start of the line.

start-y
The y coordinate of the start of the line.

dend-x
The x coordinate of the end of the line.

dend-y
The y coordinate of the end of the line.

redisplay
A boolean.

Description

The generic function move-line 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 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 \((\text{\texttt{() \texttt{()}}} \text{\texttt{)}}\) for two columns without any titles

header-args is a plist of initargs passed to the header which displays the titles of the columns. The header object is a collection. The following collection initargs are useful to pass in header-args:

:selection-callback
  The callback for clicking on the header.

:callback-type
  Defines the arguments of the selection-callback.

:items
  The items of the header object. Note that :items overrides :title if that is supplied in columns.

:print-function
  Controls how each of items is printed, providing the title of each column.

header-args may also contain the keyword :alignments. The value should be a list of alignment keywords, each of which is interpreted like an :adjust value in columns. The alignment is applied to the title only.

If auto-reset-column-widths is true, then the widths of the columns are recomputed when the items of the multi-column-list-panel are set.
Note: similar and enhanced functionality is provided by list-view.

Example

This example uses the columns initarg:

```lisp
(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:

```lisp
(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 prompt-with-list-non-focus and text-input-pane-in-place-complete.

Superclasses  interface

Description  The class non-focus-list-interface is the class of interface created and destroyed only by prompt-with-list-non-focus and text-input-pane-in-place-complete. Do not instantiate this class directly.

See also  prompt-with-list-non-focus
          text-input-pane-in-place-complete

non-focus-list-toggle-enable-filter

Function

Summary  Toggles the enabled state of the filter.

Signature  non-focus-list-toggle-enable-filter non-focus-list-interface

Arguments  non-focus-interface

A non-focus-list-interface.

Description  The function non-focus-toggle-enable-filter toggles the enabled state of the filter in a non-focus list created by prompt-with-list-non-focus or text-input-pane-in-place-complete. It has no effect if the filter is off.

It is used as the callback of the filtering-toggle.

See also  prompt-with-list-non-focus
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
non-focus-list-toggle-filter  non-focus-list-interface
non-focus-list-add-filter    non-focus-list-interface
non-focus-list-remove-filter non-focus-list-interface

Arguments
non-focus-interface
A non-focus-list-interface.

Description
These functions add or remove the filter in a non-focus list.

non-focus-list-toggle-filter calls non-focus-list-add-filter if the filter is off, otherwise it calls non-focus-list-remove-filter (it is used as the callback for the filtering-gesture).

non-focus-list-add-filter adds a filter if it is not already on, resets the text in it to empty string, and enables it.

non-focus-list-remove-filter removes the filter if it is on.

See also
prompt-with-list-non-focus

non-focus-maybe-capture-gesture

Generic Function

Summary
Maybe capture a gesture by the non-focus-interface.

Signature
non-focus-maybe-capture-gesture  non-focus-interface  gesture
=> result

Arguments
non-focus-interface
A non-focus-list-interface.
A gesture specifier.

Values

A generalized boolean.

Method Signature

\texttt{non-focus-maybe-capture-gesture} (\texttt{non-focus-interface} \texttt{non-focus-list-interface}) \texttt{gesture}

Description

The generic function \texttt{non-focus-maybe-capture-gesture} needs to return non-nil if the gesture \texttt{gesture} was captured, which means it should not be processed any more, or \texttt{nil} if \texttt{gesture} was not captured.

\texttt{gesture} should be a gesture specifier, which is an object that can be coerced to a Gesture Spec by \texttt{sys:coerce-to-gesture-spec}.

The method on \texttt{non-focus-list-interface} does the following:

1. If the gesture is \texttt{Escape} it calls \texttt{non-focus-terminate} on the non-focus window.

2. It checks whether the gesture matches any of the gestures in the \texttt{gesture-callbacks} of the window. The gesture callbacks are either explicitly defined using the initargs \texttt{:gesture-callbacks} or \texttt{:add-gesture-callbacks}, or implicitly. By default, all the gestures that are used in inplace completion (see "In-place completion" in the \textit{Lisp-Works CAPI User Guide}) are defined implicitly. These include \texttt{Up}, \texttt{Down}, \texttt{PageUp}, \texttt{PageDown} (selection in the list panel), \texttt{Return} (action), \texttt{Control+Return} and \texttt{Control+Shift+Return} (control of the filter). The implicitly defined gestures are affected by \texttt{gesture-callbacks}, \texttt{filtering-gesture} and \texttt{filtering-toggle}.

If a match is found, it is invoked as described for \texttt{gesture-callbacks} in \texttt{prompt-with-list-non-focus}.

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:
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**

\[ \text{non-focus-update ~non-focus-interface} \]

**Method Signature**

\[ \text{non-focus-update ~ (non-focus-interface ~non-focus-list-interface)} \]

**Description**
The generic function \text{non-focus-update} updates the non-focus-interface.

It has no return value.

The method on \text{non-focus-list-interface} needs to be invoked in the process in which the \text{list-updater} that was passed to \text{prompt-with-list-non-focus} is expecting to run.

It invokes the \text{list-updater} without arguments, and then updates the non-focus-interface with result. See the description of \text{list-updater} in \text{prompt-with-list-non-focus}.

Note that if \text{list-updater} returns :\text{destroy}, this invokes \text{non-focus-terminate} on the interface.

**See also**

\text{prompt-with-list-non-focus}
\text{non-focus-terminate}

---

**prompt-with-list-non-focus**

*Function*

**Summary**
Raises a non-focus window.

**Signature**

\[ \text{prompt-with-list-non-focus ~items ~&key owner ~x ~y ~choice-class}
\text{vertical-scroll ~print-function ~selection ~selected-item ~visible-items}
\text{selection-callback ~action-callback ~destroy-callback ~list-updater}
\text{gesture-callbacks ~add-gesture-callbacks ~alternative-y ~alternative-x}
\text{alternative-bottom ~alternative-right ~widget-name ~filtering-gesture}
\text{filtering-toggle ~&allow-other-keys => interface} \]
Arguments

owner A displayed CAPI pane.

x, y Integers.

alternative-x, alternative-y Integers.

alternative-bottom, alternative-right Integers or t.

choice-class A subclass of list-panel.

selection An integer.

selected-item An item.

visible-items A positive integer.

vertical-scroll A boolean.

print-function A function designator or nil.

selection-callback A function designator or nil.

action-callback A function designator or nil.

destroy-callback A function designator or nil.

list-updater A function designator or nil.

gesture-callbacks A list of pairs of the form (gesture . call-back).

add-gesture-callbacks A list of pairs of the form (gesture . call-back).

filtering-gesture A Gesture Spec.

filtering-toggle A Gesture Spec.

widget-name A string.

Values

interface A non-focus-list-interface, or nil.
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 cl:make-instance when making the list. The default value of vertical-scroll is t.

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

\text{add-gesture-callbacks} adds more gesture callbacks to those that are implicitly defined for controlling the list panel (see \text{non-focus-maybe-capture-gesture}). \text{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.

\text{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 \text{Backspace}. While the filter is visible, the list panel displays only items that match the filter. The default value of \text{filtering-gesture} is a Gesture Spec matching \text{Control+Return}.

\text{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 \text{Backspace} (when \text{non-focus-maybe-capture-gesture} is called) and passes them to the filter. When the filter is visible and disabled, characters and \text{Backspace} are captured. The default value of \text{filtering-toggle} is a Gesture Spec matching \text{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-add-filter
non-focus-list-remove-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  
ole-control-add-verbs pane menu item-identifier

Arguments
* pane  An ole-control-pane.
* menu  A menu.
* item-identifier  A string or symbol.

Description
The function `ole-control-add-verbs` adds to the menu entries for the "verbs" that the component supports. The `ole-control-pane` `pane` must have an object already, and the `menu` `menu` must have already been created, so `ole-control-add-verbs` is typically called in the `popup-callback` of `menu`.

`item-identifier` identifies an item in the menu or a component in the menu (but not in a sub-menu), either by being `eq` to the name of the item or `equalp` 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.

When the user selects an added menu item, the verb is passed to the object (by a call to `IOleObject::DoVerb`).

**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 menu
ole-control-pane

**Function**

**ole-control-close-object**

Summary  Closes the object in an `ole-control-pane`.

Signature  ole-control-close-object pane
Arguments  

```
<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pane</td>
<td>An <code>ole-control-pane</code>.</td>
</tr>
</tbody>
</table>
```

Description

The function `ole-control-close-object` closes the object that is currently in the `ole-control-pane` pane.

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

Example

See the example in

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

See also

- `ole-control-pane`

---

**ole-control-component**

*Class*

Summary

An implementation of the interfaces in the OLE Control protocol.

Package

capi

Superclasses

`com:standard-i-unknown`

Initargs

```
<table>
<thead>
<tr>
<th>Initarg</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>:pane-function</td>
<td>A function that is called when OLE embeds the Control in a container.</td>
</tr>
<tr>
<td>:create-callback</td>
<td>A function called just after the pane is created.</td>
</tr>
<tr>
<td>:destroy-callback</td>
<td>A function called just before the pane is destroyed.</td>
</tr>
</tbody>
</table>
```

Readers

- `ole-control-component-pane`
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 ole-control-pane.

Package
capi

Superclasses
pinboard-layout

Subclasses
ole-control-frame

Description
The pane class ole-control-doc can be used to implement the document around the object inside an ole-control-pane. That is, it supports the IOleInPlaceUIWindow interface. Note that this is optional, and is rarely useful.

To use it the ole-control-doc pane needs to be the parent, not necessarily directly, of an ole-control-pane. When the object calls IOleInPlaceSite::GetWindowContext, it will get (in the ppdoc [out] argument) an IOleInPlaceUIWindow interface associated with the ole-control-doc.

A ole-control-doc must have exactly one sub-pane (that is, the length of its description must be 1), but underneath this pane there can be many panes.

Normally the program does not need to do anything else with the ole-control-doc. It acts in response to resizing of the window and method calls from the object on the IOleInPlaceUIWindow interface.

Note: ole-control-doc is implemented only in LispWorks for Windows. Load the functionality by (require "embed").

Note: even though it is a subclass of pinboard-layout, normally you should not use the pinboard-layout functionality when using ole-control-doc.
Example
See the example in examples/com/ole/simple-container/doc-viewer-pair.lisp

See also ole-control-pane

ole-control-frame

Class

Summary
Implements the frame of components in an ole-control-pane.

Package
capi

Superclasses
ole-control-doc

Description
The pane class ole-control-frame implements the frame of components, that is it supports the IOleInPlaceFrame interface. When an ole-control-pane pane is created, it looks upwards in the hierarchy of panes, and if finds an ole-control-frame pane it uses this as the frame. It uses the first such pane found. When the object in the ole-control-pane calls IOleInPlaceSite::GetWindowContext, it gets back in the ppframe arg an interface associated with this frame.

Like ole-control-doc, a ole-control-frame can have only one sub-pane, which itself may contain many panes.

Normally the program does not need to do anything else with the ole-control-frame. It acts in response to resizing of the window and method calls from the object on the IOleInPlaceFrame interface.

Note that having a frame is optional, and ActiveX does not need it. It is required when embedding an application by ole-control-insert-object.

Note: ole-control-frame is implemented only in LispWorks for Windows. Load the functionality by (require "embed").

329
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

Function

ole-control-i-dispatch

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 ole-control-pane.

Signature
ole-control-insert-object pane

Arguments
pane An ole-control-pane.

Description
The function ole-control-insert-object prompts the user for a document using the Microsoft Windows function OleUIInsertObject.

When the user specifies a document in the dialog presented, ole-control-insert-object embeds this document in the ole-control-pane pane.

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

Example
See the example in examples/com/ole/simple-container/doc-viewer-pair.lisp

See also
ole-control-pane

ole-control-ole-object

Function

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

Signature
ole-control-ole-object pane => result

Arguments
pane An ole-control-pane.

Values
result A com:i-ole-object or nil.
1 CAPI Reference Entries

Description

The function `ole-control-ole-object` returns the `com:i-ole-object` (that is, the I OleObject interface) of the component of the `ole-control-pane` pane, or `nil` if there isn’t any.

**Note:** calling `ole-control-ole-object` 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-pane**

*Class*

**Summary**

A class that implements embedding of external components on Microsoft Windows.

**Package**

capi

**Superclasses**

pinboard-layout

**Initargs**

:component-name

A string or `nil`.

:user-component

A COM interface pointer or `nil`.

:save-name

A string.

:insert-callback

A function.

:close-callback

A function.

:sinks

A list of sink specifications.
Description

The class **ole-control-pane** is used to implement embedding of external components.

**Note:** **ole-control-pane** is implemented only in LispWorks for Windows. Load the functionality by `:require "embed"`.

**Note:** even though it is a subclass of **pinboard-layout**, normally you should not use the **pinboard-layout** functionality when using **ole-control-pane**.

- **component-name** (if non-nil) specifies the **component-name** of the pane, as used by **component-name**.
- **user-component** (if non-nil) is a COM interface pointer of an object that supports the **I-OLE-OBJECT** interface, and is ready to display as described in **ole-control-user-component**.
- **save-name** is used when creating the **IStorage** object for this component.
- **insert-callback** (if non-nil) is a function that takes a single argument, the pane. It is called immediately after a component was inserted into the pane. This can be used for any additional initialization that is required, for example setting the properties of the control.
- **close-callback** (if non-nil) is a function that takes a single argument, the pane. It is called just before the component is going to be closed, and can be used to do any cleanups that may be required.

**sinks** is a list of sink specifications for attaching event handlers to the source interfaces of the control. Each element of **sinks** should be a list of the form:

```
(interface-name &key invoke-callback sink-class sink)
```

The **interface-name** is used to specify the name of the source interface in the control, which is either a string naming the interface or **:default** for the default source interface. If **invoke-callback** is given, then it should be a function which will be called with the pane, method-name, method-kind and arguments vector for each source event. The **sink-class** can be
given to set the class of the internal object used for the sink interface. This is similar to calling `attach-simple-sink`. Alternatively, instead of calling `invoke-callback`, the sink can be specified directly. This is similar to calling `attach-sink`.

When the `ole-control-pane` is destroyed, the sinks are automatically detached.

There are currently three ways to insert an external component into an `ole-control-pane`. These are:

1. Call `ole-control-user-component`, which asks the user for something to insert.
2. Set the `component-name` of the pane. This can be done either via the initarg `:component-name` or by calling `(setf component-name)`.
3. Set the `user-component` of the pane, either via the initarg `:user-component` or by calling `(setf ole-control-user-component)`.

Example

```lisp
(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

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

The class `ole-control-pane-simple-sink` is used by the function `attach-simple-sink` to implement a sink interface for an embedded component on Microsoft Windows.

`ole-control-pane` is the object of type `ole-control-pane` to whose source interface the sink is being attached.

This class can be subclassed to provide additional functionality in callbacks. See `com:simple-i-dispatch` in the *LispWorks COM/Automation User Guide and Reference Manual* for more details.

*Note:* `ole-control-pane-simple-sink` is implemented only in LispWorks for Windows. Load the functionality by

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

See also

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

---

**ole-control-user-component**

*Function*

**Summary**

Gets and sets the `user-component` of an `ole-control-pane`.

**Signature**

```lisp
ole-control-user-component pane => user-component
(setf ole-control-user-component) user-component pane => user-component
```

**Arguments**

- `pane` — An `ole-control-pane`.
- `user-component` — A COM interface pointer.

**Description**

The function `ole-control-user-component` gets and sets the `user-component` of the `ole-control-pane` `pane`.

`user-component` (if non-nil) is a COM interface pointer of an object that supports the `I-OLE-OBJECT` interface, and has been opened and initialized and is ready to be displayed. This is typically created by calling `OleCreate`, `OleCreateFromFile`, `OleCreateFromData` or `OleLoad` with `pClientSite` null.
The user-component is closed and released by the ole-con-trol-pane pane, so after you have called (setf ole-control-user-component) you should not try to use it again or release it. Setting user-component also sets the pane’s component-name to nil.

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

See also ole-control-pane

option-pane

Class

Summary
A pane which offers a choice of items, but which displays only the currently selected item.

Package
capi

Superclasses choice
titled-object
simple-pane

Initargs
:enabled Non-nil if the option pane is enabled.
:visible-items-count An integer specifying the maximum length of the popup menu, or the symbol :default.
:popup-callback A function called just before the popup menu appears, or nil.
:image-function A function providing images for items, or nil.
:separator-item An item that acts as a separator between other items, or nil.
:enabled-positions

A list of fixnums, or the keyword :all.

Accessors

- option-pane-enabled
- option-pane-image-function
- option-pane-visible-items-count
- option-pane-popup-callback
- option-pane-separator-item
- option-pane-enabled-positions

Description

The class option-pane provides a pane which offers a choice between a number of items via a popup menu. Only the currently selected item is displayed.

The class option-pane inherits from choice, and so has all of the standard choice behavior such as selection and callbacks. It also has an extra enabled slot along with an accessor which is used to enable and disable the option pane.

If visible-items-count is an integer then the popup menu is no longer than this, and is scrollable if there are more items.

If visible-items-count is :default, then the popup menu is no longer than 10. This is the default value.

When popup-callback is non-nil, it should be a function of one argument that will be called just before the popup menu appears when the user clicks on it. The single argument to the function is the option pane and the return value is ignored. If required, the function can change the items or selection of the pane. The default value of popup-callback is nil.

If image-function is non-nil, it should be a function of one argument. image-function is called with each item and should return one of:

- nil  No image is shown.

An image object

The pane displays this image.
An image id or **external-image**

The system converts the value to a temporary **image** for the item and frees it when it is no longer needed.

If **image-function** is **nil**, no items have images. This is the default value.

**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
capi

Superclasses
titled-object
    simple-pane
    gp:graphics-port-mixin

Subclasses
pinboard-layout
    editor-pane
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. \texttt{:third-press} and \texttt{:nth-press} are supported only on Cocoa and Motif.

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

\begin{verbatim}
(:key [keyname] action [modifiers]*)
\end{verbatim}

where the optional \textit{keyname} is a character naming a key (no modifiers) or one of the valid Gesture Spec keywords, \textit{action} is one of \texttt{:press} or \texttt{:release} and each modifier is one of the keywords \texttt{:shift}, \texttt{:control} and \texttt{: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 \texttt{:meta} modifier will be the \texttt{Alt} key on most keyboards. On Cocoa, the \texttt{:hyper} modifier is interpreted as the \texttt{Command} key for \texttt{:key} input.

In a motion mapping, \textit{gesture} can either be defined in terms of dragging a button (in which case it is defined as a button gesture with \textit{action} \texttt{:motion}), or it can be defined for motions whilst no button is down by just specifying the keyword \texttt{:motion} with no additional arguments.

In a command mapping, \textit{gesture} should be a command which is defined using \texttt{define-command}, and provides an alias for a gesture. The following commands are predefined:

\begin{verbatim}
          (: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.
\end{verbatim}
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 nth 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: 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 capi

signature page-setup-dialog &key screen owner printer continuation

description the page-setup-dialog function displays the page setup
dialog for printer. if printer is not specified, the dialog for the
current printer is displayed.

the CAPI screen on which to display the dialog is given by
screen, which is the current screen by default.

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 func-
tion is called with the values that would normally be
returned by page-setup-dialog. On Cocoa, passing continuation
causes the dialog to be made as a window-modalsheet
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.

see also current-printer

pane-adjusted-offset

generic function

summary The pane-adjusted-offset generic function calculates the
offset required to place a pane correctly in a layout.

package capi

signature pane-adjusted-offset pane adjust available-size actual-size
&key &allow-other-keys
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

```lisp
(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

\[ \text{pane-adjusted-position \ pane \ adjust \ min-position \ max-position} \]
\[ \&\text{key} \ &\text{allow-other-keys} \]

Description

The \text{pane-adjusted-position} generic function calculates the position required by the \text{adjust} argument so that the pane \text{pane} is placed correctly within the available space in its parent layout, given a minimum and maximum position. It is a complementary function to \text{pane-adjusted-offset}, and the default method actually calls \text{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 \text{adjust}.

- \text{:top} \ \text{Place pane at the top of the region.}
- \text{:bottom} \ \text{Place pane at the bottom of the region.}
- \text{:left} \ \text{Place pane at the left of the region.}
- \text{:right} \ \text{Place pane at the right of the region.}
- \text{:center} \ \text{Place pane in the center of the region.}
- \text{(:top n)} \ \text{Place the top of pane \(n\) pixels below the top of the region.}
- \text{(:bottom n)} \ \text{Place the bottom of pane \(n\) pixels above the bottom of the region.}
- \text{(:left n)} \ \text{Place the left of pane \(n\) pixels after the left of the region.}
- \text{(:right n)} \ \text{Place the right of pane \(n\) pixels before the right of the region.}
- \text{(:center n)} \ \text{Place the center of pane \(n\) pixels below the center of the region.}

However, new methods can accept alternative values for \text{adjust} where required and can also add extra keywords. For example, \text{grid-layout} allows \text{adjust} to be a list of \text{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**  
`pane-descendant-child-with-focus pane => result`

**Arguments**  
- `pane`: A pane or layout.

**Values**  
- `result`: A pane or `nil`.

**Description**  
The function `pane-descendant-child-with-focus` attempts to find the pane inside `pane` that currently has the input focus, and returns this pane if successful. `pane-descendant-child-with-focus` may return `nil` if it does not find a pane with the focus.

**See also**  
`pane-has-focus-p`

pane-got-focus  

**Generic Function**

**Summary**  
A function called when the focus is set programmatically.

**Package**  
capi

**Signature**  
`pane-got-focus interface pane`

**Arguments**  
- `interface`: The interface of `pane`.
- `pane`: A CAPI element.

**Description**  
The generic function `pane-got-focus` is called just before the focus is set by `set-object-automatic-resize`. The supplied primary method does nothing. You may add methods on your own interface classes, which can be useful for example when the focus is set programmatically to a pane.
which is hidden inside a `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  \texttt{pane-initial-focus pane-with-children} => \texttt{pane}

Signature  \texttt{(setf pane-initial-focus) pane pane-with-children} => \texttt{pane}

Arguments  \texttt{pane-with-children}

\hspace{1cm} A pane with children.

Values  \texttt{pane}

\hspace{1cm} A child of \texttt{pane-with-children}.

Description  The generic function \texttt{pane-initial-focus} returns the child of \texttt{pane-with-children} that has the input focus when \texttt{pane-with-children} is first displayed.

\texttt{(setf pane-initial-focus)} may be used to set the initial focus pane, but only before \texttt{pane-with-children} has been created. If the setter is called after \texttt{pane-with-children} has been created, an error is signalled.

\texttt{pane-with-children} should be a pane with child panes such as a layout, an interface, a button-panel or a toolbar.

See also  \texttt{pane-has-focus-p}
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*
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.

`interface` is the top level interface of the pane. The predicate functions (those with names ending with `-p`) should return true if the operation can be performed. The other functions should perform the operations.

You can implement your own methods specializing on pane and interface classes.

**See also**  
active-pane-copy

---

### pane-popup-menu-items

**Generic Function**

**Summary**
Generates the items for the menu associated with a pane.

**Package**
capi

**Signature**
```
pane-popup-menu-items pane interface => items
```

**Arguments**
- `pane` A pane in interface `interface`.
- `interface` An interface.

**Values**
- `items` A list in which each element is a menu-item, menu-component or menu.

**Description**
The generic function `pane-popup-menu-items` generates the items for the menu associated with the pane `pane`. The default method of `make-pane-popup-menu` calls `pane-popup-menu-items` to find the items for the menu. If `pane-popup-menu-items` returns nil, then `make-pane-popup-menu` returns nil.
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))

(defun my-callback (pane)
  (capi:display-message "Callback on pane -S." pane))

(defvar edgraph)

(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)))

(defun my-callback (pane)
  (capi:display-message "Callback on pane -S." pane))

(defvar edgraph)

(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  pane  An editor-pane.

Values  text  A string.

Description  The generic function `pane-string` returns as a string the text of the buffer that is currently displayed in the editor-pane pane.

See also  editor-pane

---

**pane-supports-menus-with-images**  

*Function*

**Summary**  Tests whether a pane supports menus with images.

**Signature**  

`pane-supports-menus-with-images  pane => result`

**Arguments**  pane  A displayed CAPI pane.

**Values**  result  A boolean.

**Description**  The function `pane-supports-menus-with-images` returns `t` if the pane supports menus with images. This means that the menus display both the images and the text correctly.

See the *image-function* of *menu* for details of creating a menu with images.

When `pane-supports-menus-with-images` returns `nil`, menus can display images, but not together with text at the same item. They may also display images with transparency incorrectly.

Whether the pane supports menus with images depends on the library in which it is displayed. Support is currently limited to GTK+ and Cocoa.

See also  *menu*
parse-layout-descriptor  

**Generic Function**

**Summary**
Returns the geometry object associated with a layout’s child.

**Package**
capi

**Signature**
parse-layout-descriptor child-descriptor interface layout

**Description**
The generic function `parse-layout-descriptor` takes a description of a layout’s child, and returns the geometry object associated with that child. It is called by `interpret-description` to parse individual children in a layout.

The default method accepts a `child-desc` argument which can be a pane (subclass of `simple-pane` or `pinboard-object`), a geometry object, or a symbol naming a slot in the interface which contains such a pane.

**See also**
interpret-description  
define-layout  
layout

password-pane  

**Class**

**Summary**
The password pane is a pane designed for entering passwords, such that when the password is entered it is not visible on the screen.

**Package**
capi

**Superclasses**
text-input-pane

**Initargs**
:overwrite-character

  A base-char.

**Readers**
password-pane-overwrite-character
Description

The password pane inherits most of its functionality from text-input-pane. It starts with the initial text and caret position specified by the arguments text and caret-position respectively, and limits the number of characters entered with the max-characters argument (which defaults to nil, meaning there is no maximum).

The password pane can be enabled and disabled with the text-input-pane accessor text-input-pane-enabled.

overwrite-character is a base-char which is the character to display instead of the real characters. The default value of overwrite-character is #\*.

Example

```lisp
(setq password-pane (capi:contain
 (make-instance 'capi:password-pane
 :callback #'(lambda (password interface)
 (capi:display-message
 "Password: ~A"
 password))))))

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

(setq password-pane (capi:contain
 (make-instance 'capi:password-pane
 :max-characters 5
 :text "abc"
 :overwrite-character #\$))

(capi:password-pane-overwrite-character password-pane2)
```

See also

editor-pane
text-input-pane

play-sound

Function

Summary

Plays a loaded sound.

Package
capi
Signature  play-sound sound &key wait

Arguments  sound A sound object returned by load-sound.
           wait  A generalized boolean.

Description  The function play-sound plays the loaded sound sound.
              If wait is true then play-sound will not return until sound has
              finished playing. That is, it plays the sound synchronously.
              The default value of wait is nil.

Note: :wait t is only implemented on Microsoft Windows.

See also  load-sound
          stop-sound

pinboard-layout  Class

Summary  The class pinboard-layout provides two very useful pieces
          of functionality for displaying CAPI windows. Firstly it is a
          subclass of static-layout and so it allows its children to be
          positioned anywhere within itself (like a pinboard). Secondly
          it supports pinboard-objects which are rectangular areas
          within the layout which have size and drawing functionality.

Package  capi

Superclasses  output-pane
              static-layout

Subclasses  simple-pinboard-layout

Initargs  :highlight-style
          A keyword.
When a pinboard-layout lays out its children, it positions them at the \( x \) and \( y \) specified as hints (using \( \text{:x} \) and \( \text{:y} \)), and sizes them to their minimum size (which can be specified using \( \text{:visible-min-width} \) and \( \text{:visible-max-width} \)).

By default, the pinboard-layout is made sufficiently large to accommodate all of its children, as specified by \text{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.

\textbf{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:

- \text{:invert} Swaps the foreground and background colors.
- \text{:standard} Uses system colors.
- \text{:default} Calls draw-pinboard-object-highlighted.

The default value of highlight-style is \text{:default}.  

367
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.

```lisp
(capi:contain
 (make-instance
  'capi:pinboard-layout
 :description
 (list
  (make-instance
   'capi:image-pinboard-object
 :image
   (sys:lis pw orks-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 **pinboard-layout** with drawing capabilities.

**Package**
capi

**Superclasses**
capi-object

**Subclasses**
item-pinboard-object
image-pinboard-object
line-pinboard-object
drawn-pinboard-object
rectangle

**Initargs**
:pinboard The output pane on which the pinboard object is drawn.
:activep If t, the pinboard object is made active.

The following initargs are geometry hints, influencing the initial size and position of a **pinboard-object**:

:x The x position of the pinboard object in the pinboard.

:y The y position of the pinboard object in the pinboard.

:external-min-width
The minimum width of the pinboard object in the pinboard.

:external-min-height
The minimum height of the pinboard object in the pinboard.
The maximum width of the pinboard object in the pinboard.

The maximum height of the pinboard object in the pinboard.

The minimum visible width of the pinboard object.

The minimum visible height of the pinboard object.

The maximum visible width of the pinboard object.

The maximum visible height of the pinboard object.

The minimum width of the display region.

The minimum height of the display region.

The maximum width of the display region.

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.
keyword  A keyword denoting a Graphics Ports drawing option.

Values  value  The value of the drawing option keyword in self.

Description  The generic function pinboard-object-graphics-arg returns or sets the value of the Graphics Ports drawing option keyword in self. See the section Graphics State in the LispWorks CAPI User Guide for details of the drawing options.

See also  pinboard-object

pinboard-object-overlap-p  Generic Function

Summary  Tests whether a specified region overlaps with the region of a pinboard object.

Package  capi

Signature  pinboard-object-overlap-p  pinboard-object  top-left-x  top-left-y  bottom-right-x  bottom-right-y => result

Description  The generic function pinboard-object-overlap-p returns true if the region of the pinboard object pinboard-object overlaps with the region specified by the other arguments.

See also  pinboard-object-at-position  over-pinboard-object-p  pinboard-object  pinboard-layout
**pinboard-pane-position**  
*Generic Function*

**Summary**  
Gets and sets the location of an object inside its parent `pinboard-layout`.

**Package**  
capi

**Signature**  
`pinboard-pane-position self => x, y`

**Signature**  
`(setf pinboard-pane-position) x, y self => x, y`

**Arguments**  
`self`  
A `pinboard-object` or `simple-pane`.

**Values**  
`x, y`  
The horizontal and vertical coordinates in the `pinboard-layout` parent of `self`.

**Description**  
The generic function `pinboard-pane-position` returns as multiple values `x, y` the coordinates of `self` inside its parent `pinboard-layout`.

`(setf pinboard-pane-position)` sets the location of `self` 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

(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-confirmerv

Function

Summary
The `popup-confirmerv` 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-confirmerv 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.
message A string or nil.

modal, screen, focus, owner, x, y, and position-relative-to

These are passed to display-dialog.

title A string specifying the title of the dialog window.

title-font The font used in the title.

value-function Controls the value returned, and whether a value can be returned.

exit-function Called on exiting the dialog.

apply-function, apply-check, apply-button

Define the callback, check function and title of an Apply button.

ok-function, ok-check, ok-button

Define the callback, check function and title of an OK button.

no-button, no-function

Define the title and callback of a No button.

all-button, all-function

Define the title and callback of an All button.

cancel-button Defines the title of a Cancel button.

help-button, help-function

Define the title and callback of a Help button.

buttons Defines extra buttons.

print-function Displays ok-button, no-button, cancel-button, apply-button and all-button as button titles.

callbacks Defines callbacks for buttons.

callback-type Specifies the callback-type of buttons.

button-position One of :bottom, :top, :left, :right.
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-confirm, 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-confirm, 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 confirm-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-confirm. On Cocoa, passing continuation causes the dialog to be made as a window-modal sheet and popup-confirm 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 cl:handler-bind. The handler is established (by cl:handler-bind with type cl:error) around each callback call inside the scope of popup-con-
fimer or display-dialog. In recursive calls, only the han-
der of the innermost call to popup-confirm 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-confirm).

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-confirm (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 han-
der 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-confirm 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.

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

(capi:popup-confirm-er 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.

```lisp
(defun pane-integer (pane)
  (ignore-errors (values
    (read-from-string
      (capi:text-input-pane-text pane))))

(capi:popup-confirm-er
 (make-instance 'capi:text-input-pane
   :callback 'capi:exit-confirm-er
   :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-confirm 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-confirm
        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-confirm
 (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  

Summary  
A button with a popup menu.

Package  
capi

Superclasses  
item

Initargs  
:menu  
A menu or nil.

:menu-function  
A function designator or nil.

Accessors  
popup-menu-button-menu  
popup-menu-button-menu-function

Description  
The class popup-menu-button provides a button with a popup menu, which is displayed when the user clicks on the button.

If menu-function is non-nil, it should be function of one argument (the pane) and should return a menu object. Otherwise, menu should be a menu object.

popup-menu-button inherits from item, so you can supply text, data and so on.

Example  
See the example in capi/elements/popup-menu-button.lisp

See also  
menu

print-capi-button  

Summary  
Generates the text for a button.

Package  
capi
print-capi-button

Signature

\text{print-capi-button button} \rightarrow \text{text}

Arguments

\text{button} \quad \text{A button.}

Values

\text{text} \quad \text{A string.}

Description

The generic function \text{print-capi-button} is used to generate the text for a button.

You can add methods for your own button classes.

See also

\text{button}

\text{print-collection-item}

\text{Generic Function}

Summary

Prints an item as a string.

Package

capi

Signature

\text{print-collection-item item collection}

Arguments

\text{item} \quad \text{An item or an Lisp object.}

\text{collection} \quad \text{A collection or any Lisp object.}

Description

The generic function \text{print-collection-item} prints \text{item} as a string. It is used when \text{item} is known to be an item in \text{collection}.

An \text{item} in a collection prints using the first of these which returns non-nil: the item’s \text{text}, the item’s \text{print-function}, the collection’s \text{print-function} or the item’s \text{data}. An \text{item} not known to be in the collection is printed simply using \text{print-object}.

The method on \text{(t collection)} uses the collection’s \text{print-function}. 
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:

(defun 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

generic-function print-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.
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`.
### 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
```
The `print-file` function prints `file` to `printer`, which defaults to the current printer. If `interactive` is `t`, then a print dialog is displayed. This is the default behavior.

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

See also

- `print-editor-buffer`
- `print-text`

### print-rich-text-pane

#### Function

**Summary**

Prints the contents of a `rich-text-pane`, on Microsoft Windows.

**Package**

`capi`

**Signature**

```scheme
print-rich-text-pane pane &key jobname printer interactive selection => result
```

**Arguments**

- `pane` A `rich-text-pane`
- `jobname` A string, or `nil`.
- `printer` A printer, or `nil`.
- `interactive` A boolean.
- `selection` A boolean.

**Values**

- `result` A boolean.

**Description**

The function `print-rich-text-pane` prints the contents in `pane`.

`jobname` is the name of the print job. The default value is `nil`, meaning that the name "Document" is used.

`printer` is the printer to use. The default value is `nil`, meaning that the current-printer 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

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**
capi

**Description**
A `printer-metrics` object is returned by `get-printer-metrics`. The readers for the slots of a `printer-metrics` object are described below.

`printer-metrics-device-height` and  
`printer-metrics-device-width` 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.

`printer-metrics-dpi-x` and `printer-metrics-dpi-y` return the number of printer device units per inch in the x and y directions respectively. This typically corresponds to the printer resolution, although in some cases this may not be known. For example, a generic PostScript language compatible driver might always return 300dpi, even though it cannot know the resolution of the printer the PostScript file will actually be printed on.

`printer-metrics-height` and `printer-metrics-width` respectively return the height and width of the printable area in millimeters.

`printer-metrics-left-margin` and  
`printer-metrics-top-margin` respectively return the current left margin and current top margin of the printable area in millimeters.

`printer-metrics-max-height` and  
`printer-metrics-max-width` respectively return the greatest possible height and width of the printable area in millimeters.
printer-metrics-min-left-margin and
printer-metrics-min-top-margin respectively return the
smallest possible left margin and top margin of the printable
area in millimeters.

printer-metrics-paper-height and
printer-metrics-paper-width respectively return the
height and width of the paper selected for this printer in
millimeters.

See also get-printer-metrics

*ppd-directory*  
Variable
Summary The directory in which LispWorks looks for PPD files.
Package capi
Initial value nil
Description The variable *ppd-directory* specifies where LispWorks looks for PostScript Printer Definition (PPD) files.
This applies only on 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.
1 CAPI Reference Entries

Package       capi
Signature      printer-port-handle &optional port => handle
Arguments      port       A printer port.
Values         handle    Platform-dependent.
Description    The function printer-port-handle returns a platform-dependent value which represents the underlying handle to the printer port.

On Microsoft Windows, handle is the HDC for the printer device.

If port is passed it should be the value bound to var in with-print-job. If port is not supplied it defaults to the current printer port (dynamically bound within with-print-job).

See also       with-print-job

printer-port-supports-p Function

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

**Signature**
process-pending-messages ignored => nil

**Arguments**
The single argument is ignored.

**Description**
The function process-pending-messages processes all the pending messages in the current process, and then returns nil. 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  capi

Superclasses  range-pane  
titled-object  
simple-pane

Description  This pane is used to display progress during a lengthy task. It has no interactive behavior.

The range-pane accessors (setf range-start) and (setf range-end) are used to specify the range of values the progress bar can display.

The accessor (setf range-slug-start) is used to set the progress indication.

See also  range-pane  
titled-object

prompt-for-color  

Function

Summary  Presents a dialog box allowing the user to choose a color.

Package  capi

Signature  prompt-for-color message &key color colors owner => result, successp

Arguments  message  A string.

color  A color specification.

colors  A list.

owner  An owner window.
Values

result A color specification, or nil.

successp A boolean.

Description

The function prompt-for-color pops up a dialog box allowing the user to choose a color.

message supplies a title for the dialog on GTK+ and Motif. On Microsoft Windows message is ignored.

color provides the default color in the dialog.

colors is a list of custom color specifications that the user can choose from.

For a description of color specifications, see the "The Color System" chapter in the LispWorks CAPI User Guide.

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

**prompt-for-confirmation**

Function

Summary

Displays a dialog box with a message and Yes and No buttons.

Package
capi

Signature

prompt-for-confirmation message &key screen owner cancel-button default-button continuation => result, successp

Arguments

message A string.

screen A screen.

owner An owner window.

cancel-button A boolean.

default-button A keyword, or nil.

continuation A function or nil.
The function `prompt-for-confirmation` displays a dialog box containing `message`, with `Yes` and `No` buttons. When either `Yes` or `No` is pressed, it returns two values:

- a boolean indicating whether `Yes` was pressed
- `t` (for compatibility with other prompt functions)

`cancel-button` specifies whether a `Cancel` button also appears on the dialog. When `Cancel` is pressed, `abort` is called and the dialog is dismissed. The default value of `cancel-button` is `nil`.

`default-button` specifies which button has the input focus when the dialog appears (and is thus selected when the user immediately presses `Return`). The value `:ok` means `Yes`, the value `:cancel` means `Cancel`, and any other value means `No`. The default value of `default-button` is `nil`.

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

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

Example

```
(capi:prompt-for-confirmation "Continue?")
```
(multiple-value-bind (res success)
    (capi:prompt-for-confirmation "Yes, No or Cancel"
      :cancel-button t)
    (if success
        res
        (abort))
)

See also confirm-yes-or-no

prompt-for-directory

Function

Summary Displays a dialog prompting the user for a directory.

Package capi

Signature prompt-for-directory message &key if-does-not-exist pathname file-package-is-directory pane-args popup-args owner continuation => result, successp

Arguments

message A string.
if-does-not-exist One of :ok, :prompt or :error.
pathname A pathname, or nil.
file-package-is-directory A generalized boolean.
pane-args Arguments to pass to the pane.
popup-args Arguments to pass to the confirmer.
owner An owner window.
continuation A function or nil.

Values

result A directory pathname, or nil.
successp A boolean.
The function \texttt{prompt-for-directory} prompts the user for a directory pathname using a dialog box. Like all the prompters, \texttt{prompt-for-directory} returns two values: the directory pathname and a flag indicating success. The \texttt{successp} flag will be \texttt{nil} if the dialog was cancelled, and \texttt{t} otherwise.

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

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

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

\texttt{file-package-is-directory} is handled as by \texttt{prompt-for-file}.

\texttt{owner} specifies an owner window for the dialog. See the "Prompting for Input" chapter in the \textit{LispWorks CAPI User Guide} for details.

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

The prompt itself is created by passing an appropriate pane to \texttt{popup-confirmer}. Arguments can be passed to the \texttt{make-instance} of the pane and the call to \texttt{popup-confirmer} using \texttt{pane-args} and \texttt{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
message A string or nil.
pathname A pathname designator or nil.
ok-check A function or nil.
filter A string or nil.
filters A property list.
if-exists One of :ok or :prompt.
if-does-not-exist One of :ok, :prompt or :error.
file-package-is-directory A generalized boolean.
operation One of :open or :save.
owner An owner window.
continuation A function or nil.

Values
filename A pathname or nil.
successp A boolean.
filter-name A string.
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
("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
("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-confirm`  
`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
prompt-for-font

prompt-for-font message &key font owner => result, successp

Arguments

message  A string.
font     A font, a font description, or nil.
owner    An owner window, or nil.

Values

result   A font, or nil.
successp A boolean.

Description

The function prompt-for-font displays a dialog box allowing the user to choose a font.
message supplies a title for the dialog.
font, if non-nil, provides defaults for the dialog box. The default value is nil.
owner specifies an owner window for the dialog. See the "Prompting for Input" chapter in the LispWorks CAPI User Guide for details.

For a description of Graphics Ports fonts and font descriptions, see the LispWorks CAPI User Guide.

See also find-best-font

prompt-for-form

Function

Displays a text input pane and prompts the user for a form.

Package capi

Signature

prompt-for-form message &key package initial-value evaluate quotify ok-check value-function pane-args popup-args continuation => result, okp
Description

The function `prompt-for-form` prompts the user for a form by providing a text input pane that the form can be typed into.

The form is read in the `package` if specified or `*package*` if not. If `evaluate` is non-nil then the result is the evaluation of the form, otherwise it is just the form itself. The printed version of `initial-value` will be placed into the text input pane as a default, unless `quotify`, which defaults to `evaluate`, specifies otherwise. If `value-function` is provided it overrides the default value function which reads the form and evaluates it when required. If the `ok-check` is provided it will be passed the entered form and should return `t` if the form is a valid result.

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-form`. On Cocoa, passing `continuation` causes the dialog to be made as a window-modal sheet and `prompt-for-form` 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 calling `prompt-for-string`. Arguments can be passed to the `make-instance` of the pane and the call to `popup-confirm` 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`.

Example

Try the following examples, and each time enter `(+ 1 2)` into the input pane.

```lisp
(capi:prompt-for-form "Enter a form:"
(capi:prompt-for-form "Enter a form:" :evaluate nil)
```

See also

`prompt-for-forms`
`prompt-for-string`
`popup-confirm`
`text-input-pane`
Function

**prompt-for-forms**

**Summary**
Displays a text input pane prompting the user for a number of forms.

**Package**
capi

**Signature**
`prompt-for-forms message &key package initial-value value-function pane-args popup-args continuation => result, okp`

**Description**
The function `prompt-for-forms` 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 `package` or `*package*` if not. If `evaluate` is non-nil then the result is the evaluation of the form, else it is just the form itself.

The printed version of `initial-value` will be placed into the text input pane as a default.

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-forms`. On Cocoa, passing `continuation` causes the dialog to be made as a window-modal sheet and `prompt-for-forms` 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 a text input pane) to `popup-confirm`. Arguments can be passed to the `make-instance` of the pane and the call to `popup-confirm` using `pane-args` and `popup-args` respectively.

**Example**
Try the following example, and enter `1 2 3` into the input pane.

```lisp
(capi:prompt-for-forms "Enter some forms:")
```
See also  
prompt-for-form  
prompt-for-string  
popup-confirm  
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  
message  
A string.

min  
An integer or nil.

max  
An integer or nil.

initial-value  
An integer or nil.

ok-check  
A function or nil.

pane-args  
Arguments to pass to the pane.

popup-args  
Arguments to pass to the confirmer.

continuation  
A function or nil.

Description  
The function prompt-for-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-confirm*er. Arguments can be passed to the *make-instance* of the pane and the call to *popup-confirm*er using *pane-args* and *popup-args* respectively.

**Example**

\[
\text{(capi:prompt-for-integer "Enter an integer:" )}
\]

\[
\text{(capi:prompt-for-integer "Enter an integer:" :max 10)}
\]

\[
\text{(capi:prompt-for-integer "Enter an integer:"}
\]

\[
\text{:min 100 :max 200)}
\]

\[
\text{(capi:prompt-for-integer "Enter an integer:"}
\]

\[
\text{:ok-check 'evenp)}
\]

**See also**

*prompt-for-string*

*popup-confirm*er

*text-input-pane*

---

**prompt-for-items-from-list**

*Function*

**Summary**

Prompts with a choice of items.

**Package**

capi
prompt-for-items-from-list

**Signature**

`prompt-for-items-from-list items message &key pane-args popup-args interaction choice-class continuation => result, successp`

**Arguments**

- `items` A sequence.
- `message` A string.
- `pane-args` Arguments to pass to the pane.
- `popup-args` Arguments to pass to the confirmer.
- `choice-class` A class name.
- `continuation` A function or `nil`.

**Description**

The function `prompt-for-items-from-list` is similar to `prompt-with-list`. `interaction` defaults to `:extended-selection`.

**See also**

`prompt-with-list`

---

**prompt-for-number**

**Function**

**Summary**

Prompts the user for a number.

**Package**

`capi`

**Signature**

`prompt-for-number message &key min max initial-value ok-check pane-args popup-args continuation => result, successp`

**Arguments**

- `message` A string.
- `min` A number or `nil`.
- `max` A number or `nil`.
- `initial-value` A number or `nil`.
prompt-for-number

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 okp.

ok-check A function or nil.
pane-args Arguments to pass to the pane.
popup-args Arguments to pass to the confirmer.
continuation A function or nil.
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-confirm`. Arguments can be passed to the call to `popup-confirm` 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

```lisp
(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-confirm` 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

```lisp
(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:

```lisp
(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-confirm`
- `text-input-pane`
**prompt-for-value**

**Function**

**Summary**
Prompts the user for a form to evaluate.

**Package**
capi

**Signature**
prompt-for-value message &key package initial-value value-function pane-args popup-args continuation

**Description**
The function `prompt-for-value` prompts the user for a form and returns the result of evaluating that form.

The form is read in the `package` if specified or `*package*` if not and the result is the evaluation of the form.

If `initial-value` is supplied it provides a default form.

If `value-function` is supplied it overrides the default value function which reads the form and evaluates it.

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-value`. On Cocoa, passing `continuation` causes the dialog to be made as a window-modal sheet and `prompt-for-value` 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 a `text-input-pane` to `popup-confirm`. Arguments can be passed to the `make-instance` of the pane and the call to `popup-confirm` using `pane-args` and `popup-args` respectively.

**Example**

(capi:prompt-for-value
 "Square"
 :initial-value '(+ 1 2 3)
 :value-function
 #'(lambda (text)
   (let ((res (eval (read-from-string text))))
     (* res res))))
prompt-with-list

Function

Summary
Prompts the user to select an item or items from a choice.

Package
capi

Signature
prompt-with-list items message &key choice-class interaction value-function pane-args popup-args continuation => result, successp

Arguments
items A sequence.
message A string.
choice-class A class name.
value-function A function, or nil.
pane-args Arguments to pass to the pane.
popup-args Arguments to pass to the confirmer.
continuation A function or nil.

Description
The function prompt-with-list prompts the user with a choice. The user’s selection is normally returned by the prompter.

items supplies the items of the choice.
message supplies a title for the choice.
choice-class determines the type of choice used in the dialog. choice-class defaults to list-panel, and must be a subclass of choice.
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-confirm. Arguments can be passed to the make-instance of the pane and the call to popup-confirm 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-confirm
- 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
message A string.
owner An owner window, or nil.
continuation A function or nil.

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

```lisp
(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
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**

```lisp
(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** capi

**Signature** `quit-interface pane &key force => result`

**Arguments**
- `pane` A CAPI element.
- `force` A boolean. The default value is `nil`.

**Values**
- `result` `t` if the interface was closed, `nil` otherwise.

**Description** The function `quit-interface` closes the top level interface containing `pane`, but first it verifies that it is okay to do this by calling the interface’s `confirm-destroy-function`. If it is OK to close the interface, it then calls `destroy` to do so. If `force` is true, then neither the `confirm-destroy-function` or the `destroy-callback` are called, and the window is just closed immediately.

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

**Example** Here are two examples demonstrating the use of `quit-interface` with the `destroy-callback` and the `confirm-destroy-function`.

```lisp
(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

```lisp
(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

range-start
range-end
range-slug-start
range-slug-end
range-callback
range-orientation

Description

The class range-pane exists to support the progress-bar and slider classes. Consult the reference pages for progress-bar and slider for further information.

See also

progress-bar
slider

range-set-sizes

Function

Summary

Set values in a range-pane.

Signature

range-set-sizes range-pane &key start end slug-start slug-end redisplay

Arguments

range-pane A range-pane.
start A real number or nil.
end A real number or nil.
slug-start A real number or nil.
slug-end A real number or nil.

Description

The function range-set-sizes set the values in the range-pane range-pane for any value of start, end, slug-start or slug-end that is supplied as non-nil.

For each of start, end, slug-start and slug-end, if the value is nil or not supplied, the corresponding value in range-pane is not changed.

If redisplay is true then range-pane is redisplayed with the new values.
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: :filled  A boolean, default value nil.

Accessors: filled

Description: The class rectangle provides a simple pinboard-object that draws a rectangle.

filled determines whether the rectangle is filled.

redisplay-collection-item  Generic Function

Summary: Redisplays the area in a collection that belongs to an item.

Package: capi

Signature: redisplay-collection-item collection item

Description: The generic function redisplay-collection-item redisplays item in collection.

There are methods supplied for graph-pane and tree-view.

See also: collection

redisplay-interface  Generic Function

Summary: Updates the state of an interface.

Package: capi

Signature: redisplay-interface interface
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-confirm` 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  

**Summary**  Redraws any pinboard objects within a specified rectangle.

**Package**  capi

**Signature**  

```
redraw-pinboard-layout pinboard x y width height
&optional redisplay
```

**Description**  The function `redraw-pinboard-layout` causes any pinboard objects within the given rectangle of the pinboard layout to get redrawn.

If `redisplay` is `nil`, then the redisplay will be cached until a later update. The default for `redisplay` is `t`.

**See also**  

- pinboard-object
- redraw-pinboard-object

redraw-pinboard-object  

**Summary**  Redraws a specified pinboard object.

**Package**  capi

**Signature**  

```
redraw-pinboard-object object &optional redisplay
```

**Description**  The function `redraw-pinboard-object` causes the pinboard object `object` to be redrawn, unless `redisplay` is `nil` in which case the redisplay will be cached until a later update. The default for `redisplay` is `t`.

**Example**  There are examples in the directory `examples/capi/graphics/`.  

437
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

```lisp
(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

capi

**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 replacable dialog.

**Package**

capi

**Signature**

`replace-dialog interface &rest args => nil`

**Arguments**

- `interface` An interface.
- `args` Other arguments as for `display-dialog`.

**Description**

The function `replace-dialog` displays a dialog in the same way the `display-dialog` does, except that it also destroys the existing 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.

`replace-dialog` displays the dialog like `display-dialog`. 
replace-items

Generic Function

Summary
Repases some items in a collection.

Package
capi

Signature
replace-items collection items &key start new-selection

Arguments

*collection* A collection.

*items* A list.

*start* A non-negative integer.

*new-selection* A list specifying the selection.

Description
The generic function replace-items replaces some items in the *collection*. 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**

```lisp
(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 dis-
ables 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**
capi

**Signature**
rich-text-pane-character-format pane &key selection => result

**Arguments**
- pane: A rich-text-pane.
- selection: A boolean.

**Values**
- result: A plist.

**Description**
The function rich-text-pane-character-format returns as a plist the current character attributes for pane.
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

\[
\text{(getf (capi:rich-text-pane-character-format pane) :bold :unknown)}
\]

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**  
capi

**Signature**  
\[
\text{rich-text-pane-operation pane operation \&rest args => result, result2}
\]

**Arguments**  
- pane  
  A rich-text-pane.
- operation  
  A keyword specifying the operation to perform.
args

The value or values to use, when the operation is setting something.

Values

result

Various, see below.

result2

Returned only for operation :get-selection, see below.

Description

The valid values of operation on Microsoft Windows and Cocoa are:

:pastep, :cutp or :copyp

result is a boolean indicating whether it is currently possible to perform a :paste, :cut or :copy operation.

:paste, :cut, or :copy

Performs the indicated operation.

:select-all

Selects all the text.

:set-selection

args should be two integers start and end.
Sets the selection to the region bounded by start (inclusive) and end (exclusive).

:get-selection

Returns as multiple values the bounding indexes of the selection. result is the start (inclusive) and result2 is the end (exclusive). If there is no selection, both values are the index of the insertion point.

:can-undo or :can-redo

result is a boolean indicating whether it is currently possible to perform an :undo or :redo operation.

:undo

Undoes the last editing operation. Note that, after typing, it is the whole input, rather than a single character, that is undone. The
:undo operation may be repeated successively, to undo previous editing operations in turn.

Note: with RichEdit 1.0, :undo does not work repeatedly - it only undoes one previous editing operation. See rich-text-version.

:redo
Undoes the effect of the last :undo operation. The :redo operation may be repeated successively, to cancel the effect of previous :undo operations in turn.

Note: with RichEdit 1.0, :redo does not work. See rich-text-version.

:get-modified
result is the value of a boolean modified flag. This flag can be set by the :set-modified operation. Also, editing the text sets it to true.

:set-modified
Sets the modified flag. The argument is a boolean.

:save-file
Saves the text to a file. Details below.

:load-file
Loads the text from a file. Details below.

Additionally these values of operation are valid on Microsoft Windows, only:

:get-word-wrap
Returns a value indicating the word wrap, which can be the keyword :none. result can also be the keyword :window or a CAPI printer object, meaning that the text wraps according to the width of the window or the printer.
:set-word-wrap

Sets the word wrap. The argument can be as described for :get-word-wrap, and additionally it can be the keyword :printer, meaning the current-printer.

:hide-selection

Specifies whether the selection should be hidden (not highlighted) when pane does not have the focus. The argument is a boolean.

For operations :save-file and :load-file, args is a lambda list

filename &key selection format plain-text

filename is the file to save or load.

selection is a boolean, with default value nil.

format is nil or a keyword naming the file format. Values include :rtf and :text meaning Rich Text Format and text file respectively.

plain-text is a boolean, with default value nil.

With operation :save-file, if selection is true, only the current selection is saved. If selection is nil, all the text is saved. The default value of format is :rtf and there are two further allowed values, :rtfnoobjs and :textized. These are like :rtf and :text except in the way they deal with COM objects. See the documentation for SF_RTFNOOBS and SF_TEXTIZED in the EM_STREAMOUT entry in the MSDN for details. When saving with format :rtf or :rtfnoobjs, if plain-text is true, then keywords that are not common to all languages are ignored. With other values of format, plain-text has no effect.

With operation :load-file, if selection is true, the unselected text is preserved. If there is a selection, the new text replaces it. If there is no selection, the new text is inserted at the cur-
rent insertion point. If selection is nil, all the text is replaced. The default value of format is nil, meaning that the RTF signature is relied upon to indicate a Rich Text Format file. If plain-text is true, then keywords that are not common to all languages are ignored.

Example

```lisp
(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
```

**Function**

**rich-text-pane-paragraph-format**

**Summary**
Returns the paragraph format.

**Package**
capi

**Signature**
rich-text-pane-paragraph-format pane => result

**Arguments**
pane A rich-text-pane.
Values  

| result | A plist. |

Description  
The function `rich-text-pane-paragraph-format` returns as a plist the paragraph attributes of the current paragraphs in `pane`.

For the possible attributes, see `set-rich-text-pane-paragraph-format`.

See also  
`rich-text-pane`

### rich-text-version

*Function*

Summary  
Identifies the version of RichEdit in use, on Microsoft Windows.

Package  
capi

Signature  
`rich-text-version => result`

Values  

| result | A keyword indicating the version of the RichEdit control in use. |

Description  
`result` is `:rich-edit-2.0` if RichEdit 2.0 or newer is loaded. Otherwise `result` is `:rich-edit-1.0`.

`rich-text-version` is supported only on Microsoft Windows.

See also  
`rich-text-pane`

### right-angle-line-pinboard-object

*Class*

Summary  
A subclass of `pinboard-object` that displays a line drawn around two edges of the area enclosed by the pinboard object.
### Package

capi

### Superclasses

line-pinboard-object

### Initargs

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>:type</td>
<td>The type of line.</td>
</tr>
</tbody>
</table>

### Description

A subclass of `line-pinboard-object` which displays a line around the edge of the pinboard object rather than diagonally.

*type* can be one of two values.

- **:vertical-first**
  
  Draw top-left to bottom-left to bottom-right.

- **:horizontal-first**
  
  Draw top-left to top-right to bottom-right.

The main use of this class is to produce graphs with right-angled edges rather than diagonal ones.

### Example

```lisp
(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))
```

### See also

pinboard-layout

### row-layout

#### Class

**Summary**

The row-layout class lays its children out in a row.
Package        capi

Superclasses  grid-layout

Initargs    :ratios    The size ratios between the layout’s children.
            :adjust   The vertical adjustment for each child.
            :gap      The gap between each child.
            :uniform-size-p
                    If t, each child in the row has the same width.

Accessors   layout-ratios

Description This 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 argu-
ments x-ratios, y-adjust, x-gap and x-uniform-size-p.
description may also contain the keywords :divider and
:separator which automatically create a divider or separator
as a child of the row-layout. The user can move a divider, but
cannot move a separator.

When specifying :ratios in a row with :divider or :sepa-
ratror, you should use nil to specify that the divider or separ-
ator is given its minimum size.

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

```lisp
(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:

(setq row (capi:contain
    (make-instance 'capi:row-layout
 :description
 (list (make-instance 'capi:push-button
 :text "foo")
 nil
 (make-instance 'capi:push-button
 :text "bar")
 :adjust :center))

See also
column-layout

screen

Class

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

Package
capi
Superclasses  
capi-object

Subclasses  
color-screen  
mono-screen

Initargs  
;width  The width in pixels of the screen.
;height  The height in pixels of the screen.
;number  The screen number.
;depth  The number of color planes in the screen.
;interfaces  A list of all of the interfaces visible on the screen.

Readers  
screen-width  
screen-height  
screen-number  
screen-depth  
screen-interfaces  
screen-width-in-millimeters  
screen-height-in-millimeters

Description  
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 screen 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.

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.

On Motif, there is one CAPI screen for each X11 screen.

Compatibility note  
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.

Example  
(setq screen (capi:convert-to-screen))  
(capi:screen-width screen)
(capi:screen-height screen)

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

(capi:screen-interfaces screen)

See also convert-to-screen

screen-active-interface 

Function
Summary
Returns the active interface on a screen.
Package capi
Signature
screen-active-interface screen => interface
Arguments
screen A screen or document-container
Values
interface An interface, or nil.
Description
The function screen-active-interface returns the currently active interface on the screen screen, or nil if no CAPI interface is active or if this cannot be determined.

screen-active-interface also works with document-container, returning the active interface within the container.

See also document-container
screen

screen-active-p 

Function
Summary
Determines whether a screen is active.
Package capi
**screen-active-p**

**Signature**

```
(screen-active-p screen => result)
```

**Arguments**

`screen`  
A screen.

**Values**

`result`  
A boolean.

**Description**

The function `screen-active-p` is the predicate for whether a screen is active.

**See also**

`screen`

---

**screen-logical-resolution**

**Function**

**Summary**

Returns the logical resolution of `screen`.

**Package**

`capi`

**Signature**

```
(screen-logical-resolution screen => xlogres, ylogres)
```

**Arguments**

`screen`  
A screen.

**Values**

`xlogres, ylogres`  
Integers representing the logical resolution of `screen` in DPI.

**Description**

The function `screen-logical-resolution` returns the logical resolution of `screen`, as dots per inch in the x and y directions.

**See also**

`screen`

---

**screen-internal-geometry**

**Function**

**Summary**

Returns the geometry of the usable region of a screen or document container.

**Package**

`capi`
Signature  
\texttt{screen-internal-geometry screen} \Rightarrow \texttt{x, y, width, height}

Arguments  
\texttt{screen} \hspace{1cm} \textnormal{A screen.}

Values  
\texttt{x} \hspace{1cm} \textnormal{An integer.}
\texttt{y} \hspace{1cm} \textnormal{An integer.}
\texttt{width} \hspace{1cm} \textnormal{An integer.}
\texttt{height} \hspace{1cm} \textnormal{An integer.}

Description  
The function \texttt{screen-internal-geometry} returns the geometry (as \texttt{x, y, width} and \texttt{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 \texttt{screen-internal-geometry} works with \texttt{document-container}, returning the current size of the container (which may vary over time).

See also  
\texttt{document-container}
\texttt{screen}

\texttt{screens}  
\textbf{Function}

Summary  
Returns the active screens for a library.

Package  
\texttt{capi}

Signature  
\texttt{screens \&optional library} \Rightarrow \texttt{result}

Arguments  
\texttt{library} \hspace{1cm} \textnormal{A library name, a list, or :any.}

Values  
\texttt{result} \hspace{1cm} \textnormal{A list.}

Description  
The function \texttt{screens} returns as a list all the active screens for \texttt{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**

capi

**Superclasses**

range-pane  
simple-pane  
titled-object

**Initargs**

- `:line-size`  
  The distance scrolled by the scroll-line gesture.

- `:page-size`  
  The distance scrolled by clicking inside the scroll bar.

- `:callback`  
  A function called after a scroll gesture, or `nil`.

**Accessors**

- `scroll-bar-line-size`
- `scroll-bar-page-size`

**Description**

The class *scroll-bar* 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 *simple-pane*.

`line-size` 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 *line-size* is 1.

`page-size` is the logical size of a page, and is the distance moved when the user clicks inside the scroll bar. The default value of *page-size* is 10.

`callback` can be `nil`, meaning there is no callback. This is the default value. Otherwise, is a function of four arguments, the interface containing the scroll-bar, the scroll-bar itself, the mode of scrolling and the amount of scrolling. It has this signature:
callback interface scroll-bar how where

how can be one of :line, :page, :move, or :drag.

If how is :line, then where is an integer indicating how many lines were scrolled.

If how is :page, then where is an integer indicating how many pages were scrolled.

If how is :move or :drag, then where is an integer giving the new location of the slug-start, or :start or :end.

Note: the location of the slug can be found by the range-pane accessor range-slug-start.

Note: Rather than using scroll-bar, it is more usual to add scroll bars to a pane by the simple-pane initargs :horizontal-scroll and :vertical-scroll

Example

(defun sb-callback (interface sb how where)
  (declare (ignorable interface))
  (format t "Scrolled ~a where ~a : ~a~%
          how where (range-slug-start sb)))

(contain
  (make-instance 'capi:scroll-bar
    :callback 'sb-callback
    :page-size 10
    :line-size 2
    :visible-min-width 200))

See also simple-pane

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

Value:

- One of `t`, `nil` or `:non-mouse`.

Method Signature

- `scroll-if-not-visible-p simple-pane`

(setf `scroll-if-not-visible-p`) `value` `simple-pane`

Description

The generic function `scroll-if-not-visible-p` accesses the `scroll-if-not-visible-p` attribute of a pane.

The value of this attribute has these meanings:

- `t`:
  When `pane` 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.

- `nil`:
  Never scroll the parent to make a pane visible.

- `:non-mouse`:
  Like `t`, except that it does not scroll when the focus is given as a result of a mouse click in `pane`.

`scroll-if-not-visible-p` is called by CAPI each time it may need to scroll the parent. The method on `simple-pane` returns a value that is kept internally, and can be set by the default setf method.

You can specialize `scroll-if-not-visible-p` on your classes, but note that it is called often when the user clicks on any pane, so it must be reasonably fast.

The setter sets the `scroll-if-not-visible-p` attribute. It is called when the initarg `:scroll-if-not-visible-p` is used in making a `simple-pane` (or a subclass) instance, and can be called by your program. `value` must be `t`, `nil` or `:non-mouse`.

The method on `simple-pane` sets the internal value that is used by `scroll-if-not-visible-p` on `simple-pane`.

See also

- `simple-pane`
search-for-item

Generic Function

Summary
The generic function search-for-item returns the index of an item in a collection.

Package
capi

Signature
search-for-item collection item

Description
Returns the index of item in the collection, using the collection-test-function to determine equality, and returns nil if no match is found.

search-for-item is the counterpart function to get-collection-item which given an index, finds the appropriate item.

See also
get-collection-item
collection

selection

Function

Summary
Returns the primary selection.

Package
capi

Signature
selection self &optional format => result

Arguments
self A displayed CAPI pane or interface.
format A keyword.

Values
result A string, an image, a Lisp object, or nil.

Description
The function selection returns the contents of the primary selection as a string, or nil if there is no selection.

format controls what kind of object is read. The following values of format are recognized:
The object is a string. This is the default value.

:image
The object is of type image, converted from whatever format the platform supports.

:value
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**
capi

**Signature**

```
selection-empty self &optional format => result
```

**Arguments**

- `self`: A displayed CAPI pane or interface.
- `format`: A keyword.

**Values**

- `result`: t or nil.
The function `selection-empty` returns `nil` if there is a primary selection of the kind indicated by `format`, or `t` if there is no such selection.

`format` controls what kind of object is checked. The following values of `format` are recognized:

- `:string` The object is a string. This the default value.
- `:image` The object is of type `image`, converted from whatever format the platform supports.
- `:value` The object is the Lisp value.

See also `image selection`

---

**set-application-interface**

*Function*

**Summary**

Specifies the main Cocoa application interface.

**Package**

capi

**Signature**

`set-application-interface interface`

**Arguments**

`interface` An object of type `cocoa-default-application-interface`

**Description**

The function `set-application-interface` sets `interface` as the main application interface. This interface is used to supply the application menu and receives various callbacks associated with the application.

`set-application-interface` must be called before any CAPI functions that make the `screen` object (such as `convert-to-screen` and `display`).

`interface` should not be displayed like a normal interface.
**set-application-interface** is only applicable when running under Cocoa.

See also **cocoa-default-application-interface**

---

**set-button-panel-enabled-items**

*Generic Function*

**Summary**

Sets the enabled state of the items in a button panel.

**Package**

capi

**Signature**

```
set-button-panel-enabled-items button-panel &key enable disable set test key
```

**Description**

The generic function **set-button-panel-enabled-items** sets the enabled state of the items in a button panel. If **set** is t, then **enable** is ignored and all items are enabled except those in the **disable** list. If **set** is nil, **disable** is ignored and all items are disabled except those in the **enable** list. If **set** is not given, the items in the **enable** list are enabled and the items in the **disable** list are disabled. If an item is in both lists, it is enabled. A button is in a list when the data of the button matches one of the items in the list. A match is defined as a non-nil return value from the test function. The default test function is **equal**.

See also **button-panel**

**redisplay-interface**

---

**set-clipboard**

*Function*

**Summary**

Sets the contents of the system clipboard.

**Package**
capi
Signature  

\texttt{set-clipboard \textit{self} \textit{value} \&optional \textit{string} \textit{plist} => \textit{result}}

Arguments  

\textit{self}  
A displayed CAPI pane or interface.

\textit{value}  
A Lisp object (not necessarily a string) to make available within the local Lisp image.

\textit{string}  
The string representation of \textit{value} to export, or \texttt{nil}. If \texttt{nil} and \textit{value} is a string, then that will be exported as the string.

\textit{plist}  
A property list of additional format/value pairs to export. The currently supported formats are as described for \texttt{clipboard}. You can export more than one format simultaneously.

Values  

\textit{result}  
A string, or \texttt{nil}.

Description  
The function \texttt{set-clipboard} sets the contents of the system clipboard to be the text of \textit{string}.

In Microsoft Windows applications (including LispWorks in Windows emulation mode), the contents of the system clipboard is usually accessed by the user with the `Ctrl+V` gesture.

The X clipboard can be accessed by the `Ctrl+V` gesture in KDE/Gnome emulation, or by running the program \texttt{xclipboard} or the Emacs function \texttt{x-get-clipboard}. The most likely explanation for apparent inconsistencies after \texttt{set-clipboard} 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 `Command+V` gesture.

Example  
To export an image:

\texttt{(capi:set-clipboard pane nil nil (list :image image))}
To export an image with a text description

```lisp
(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
**set-default-editor-pane-blink-rate**

**Arguments**

`blink-rate` A non-negative real number, or nil.

**Description**

The function `set-default-editor-pane-blink-rate` sets the default to use for the editor pane cursor blinking rate. This default value is used when `editor-pane-blink-rate` returns nil.

Initially the setting is if this call has been made:

```
(set-default-editor-pane-blink-rate nil)
```

This means that the native blink rate will be used.

The argument `blink-rate` is interpreted as a blinking rate as described in `editor-pane-blink-rate`.

**See also**

`editor-pane-blink-rate`

`editor-pane-native-blink-rate`

---

**set-default-interface-prefix-suffix**

**Function**

**Summary**

Sets the default suffix and prefix that are added to each interface title.

**Package**

capi

**Signature**

```
(set-default-interface-prefix-suffix &key prefix suffix
  child-prefix child-suffix => prefix, suffix, child-prefix, child-suffix)
```

**Arguments**

- `prefix` A string or nil.
- `suffix` A string or nil.
- `child-prefix` A string or nil.
- `child-suffix` A string or nil.

**Values**

- `prefix` A string or nil.
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.
set-geometric-hint

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

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
capi

Signature
set-object-automatic-resize object &key x-align y-align x-offset y-offset x-ratio y-ratio width-ratio height-ratio aspect-ratio aspect-ratio-y-weight pinboard

Arguments

object A pinboard-object or a simple-pane.
x-align nil, :left, :center or :right.
y-align nil, :top, :center or :bottom.
x-offset A real number, default value 0.
y-offset A real number, default value 0.
x-ratio A positive real number or nil.
y-ratio A positive real number or nil.
width-ratio A positive real number or nil.
height-ratio A positive real number or nil.
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

\[
\frac{\text{calculated-height}}{\text{calculated-width}} \times \text{aspect-ratio}
\]

3. Compute correction as

\[
\text{correction} = \left(\text{expt aspect-ratio-ratio aspect-ratio-y-weight}\right)
\]

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:
(set-object-automatic-resize
  object
  :x-align :center :y-align :center
  :x-ratio 0.5 :y-ratio 0.5
  :aspect-ratio 0.6 :width-ratio 0.1)

See also
  manipulate-pinboard
  pinboard-layout
  pinboard-object
  simple-pane

set-pane-focus  

Generic Function

Summary  Sets the input focus to a pane.

Package  capi

Signature  set-pane-focus pane

Arguments  pane  An instance of a subclass of simple-pane or choice.

Description  The function set-pane-focus sets the input focus to pane or one of its children.

See also  pane-has-focus-p

set-rich-text-pane-character-format  

Function

Summary  Sets the character format.

Package  capi

Signature  set-rich-text-pane-character-format pane &key selection
  attributes-plist => result
Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pane</td>
<td>A rich-text-pane.</td>
</tr>
<tr>
<td>selection</td>
<td>A boolean.</td>
</tr>
<tr>
<td>attributes-plist</td>
<td>A plist or :default.</td>
</tr>
</tbody>
</table>

Values

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

Description

The function `set-rich-text-pane-character-format` sets current character attributes for `pane`.

`selection` determines the text for which the attributes are set. If `selection` is `nil`, then the attributes are set on the next text entered in `pane`. If `selection` is `t`, then the attributes are set on the current selection. The default value of `selection` is `t`.

If `attributes-plist` is the symbol `:default` then the default character format of the pane is used. Otherwise `attributes-plist` is a plist of keywords and values. These are the valid keywords on Microsoft Windows and Cocoa:

- `:bold` A boolean.
- `:italic` A boolean.
- `:underline` A boolean.
- `:face` A string naming a font.
- `:color` A color spec or alias specifying the foreground color.
- `:size` The size of the font.

Additionally these `attributes-plist` keywords are valid on Microsoft Windows only:

- `:strikeout` A boolean.
- `:offset` An integer specifying the vertical offset of characters from the line (a positive value makes them superscript and a negative value makes them subscript).
- `:protected` A boolean.
:charset  A cons (charset , pitch-and-family) where
charset has the value of a Microsoft Windows
charset identifier, and pitch-and-family is the
value of (logior pitch family) where pitch
and family have the value of a Windows
pitch and a Windows font family respec-
tively.

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 , a little longer, a little longer
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:
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**
capi

**Signature**

```
(set-rich-text-pane-paragraph-format pane attributes-plist) => result
```

**Arguments**

*pane*  
A rich-text-pane.

*attributes-plist*  
A plist, or :default.

**Values**

*result*  
A plist.

**Description**
The function `set-rich-text-pane-paragraph-format` sets paragraph attributes for the current paragraphs in `pane`.

The current paragraphs are those paragraphs which overlap the current selection, or the paragraph containing the insertion point if there is no selection.

If `attributes-plist` is the symbol :default then the default paragraph format of the `pane` is used. Otherwise `attributes-plist` is a plist of keywords and values. These are the valid keywords on Microsoft Windows and Cocoa:

* :alignment  :left, :right or :center.
* :start-indent  A number setting the indentation.
* :offset-indent  A number modifying the indentation.
: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 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

```lisp
(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**

- `self` : A displayed CAPI pane or interface.
- `value` : A Lisp object (not necessarily a string) to make available within the local Lisp image.
- `string` : The string representation of `value` to export, or `nil`. If `nil` and `value` is a string, then that will be exported as the string.
- `plist` : A property list of additional format/value pairs to export. The currently supported formats are `:string`, 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 result A string, or nil.

Description The function set-selection sets the primary selection to be
the text of string.

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. The most likely
explanation for apparent inconsistencies after set-selection
is that the pasting application doesn’t use the primary
selection. For instance, Emacs is configurable by the variable
interprogram-paste-function.

See also selection
set-clipboard

set-printer-metrics Function

Summary Sets the metrics in the given printer.

Package capi

Signature set-printer-metrics printer &key left-margin top-margin
width height

Description The function set-printer-metrics 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.

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  

```
set-text-input-pane-selection pane start end
```

Arguments  

- **pane**: A text-input-pane.
- **start, end**: Bounding indexes for a subsequence of the text of `pane`.

Description  
The function `set-text-input-pane-selection` sets the selection in `pane` to be the text bounded by the indexes `start` (inclusive) and `end` (exclusive).

See also  

- `text-input-pane-selection`
- `text-input-pane`

---

**set-top-level-interface-geometry**  

*Generic Function*

Summary  
Sets the geometry of a top level interface.

Package  
`capi`

Signature  

```
set-top-level-interface-geometry interface &key x y width height
```

Arguments  

- **interface**: A CAPI interface.
- **x, y, width, height**: Integers specifying the new geometry.

Description  
The coordinates of `interface` are modified according to the keyword arguments passed. The value of `interface` 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.
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-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

shell-pane-command

Description
The class shell-pane creates an editor in which a subprocess runs.

User input is interpreted as input to the subprocess. In particular, when the user enters Return in the last line, the line is sent to the subprocess. The output of the subprocess is displayed in the pane.

The default value of command is nil, which means that the actual command is determined as follows:

On Unix/Linux and Mac OS X, the value of the environment variable $ESHELL$ is used if set, and otherwise the environment variable $SHELL$ is consulted. If that is not set, then /bin/csh (/bin/sh on SVR4 platforms) is run.

On Microsoft Windows 98/ME, command is run.

On Windows 2000/XP/Vista/Windows 7, cmd is run.

Example
This function emulates user input on pane:

(defun send-keys-to-pane-aux (pane string newline-p)
  (loop for char across string
        do (capi:call-editor pane char))
  (if newline-p
      (capi:call-editor pane #\Return))))

This function trampolines to send-keys-to-pane-aux on the right process:

(defun send-keys-to-pane (pane string newline-p)
  (capi:apply-in-pane-process pane
   'send-keys-to-pane-aux
   pane string newline-p))

(setq sp (capi:contain
           (make-instance 'capi:shell-pane
                           :visible-min-width
                           '(character 60)
                           :visible-min-height
                           '(character 30))))
This call emulates the user typing dir followed by Return:

(send-keys-to-pane sp "dir" t)

show-interface  

Summary  The show-interface function brings the interface containing a specified pane back onto the screen.

Package  capi

Signature  show-interface pane

Description  This brings the interface containing pane back onto the screen. To hide it again, use hide-interface.

See also  hide-interface  activate-pane  interface

show-pane  

Summary  Restores the specified pane to the screen.

Package  capi

Signature  show-pane pane => pane

Arguments  pane  An instance of simple-pane or a subclass.

Description  The function show-pane restores the pane pane to the screen if it is hidden (for instance by hide-pane) or iconified.

See also  hide-pane  show-interface
**simple-layout**

*Class*

**Summary**

A simple-layout is a layout with a single child, and the child is resized to fill the space (where possible).

**Package**

capi

**Superclasses**

x-y-adjustable-layout

**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**

```lisp
(capi:contain (make-instance
               'capi:simple-layout
               :description (list (make-instance
                                   'capi:text-input-pane))))
```

**See also**

layout
row-layout
column-layout

**simple-network-pane**

*Class*

**Summary**

A graph pane which arranges its nodes in a grid.

**Package**

capi

**Superclasses**

graph-pane

**Initargs**

: x-gap The horizontal node spacing.

: y-gap The vertical node spacing.

**Description**

The class simple-network-pane provides a graph which lays out its nodes in a rectangular grid by a simple algorithm.
The default values of \textit{x-gap} and \textit{y-gap} are 200 and 100 respectively.

\texttt{simple-network-pane} is a subclass of \texttt{choice}, so for details of its selection handling, see \texttt{choice}.

\textbf{Example} See the file \texttt{examples/capi/graphics/network.lisp}.

\section*{simple-pane \hspace{1cm} Class}

\textbf{Summary} The class \texttt{simple-pane} is the superclass for any elements that actually appear as a native window, and is itself an empty window.

\textbf{Package} \texttt{capi}

\textbf{Superclasses} \texttt{element}

\textbf{Subclasses} \texttt{display-pane}
\texttt{interface}
\texttt{title-pane}
\texttt{button-panel}
\texttt{list-panel}
\texttt{option-pane}
\texttt{output-pane}
\texttt{progress-bar}
\texttt{slider}
\texttt{text-input-pane}
\texttt{tree-view}
\texttt{toolbar}
\texttt{layout}
\texttt{button}

\textbf{Initargs} \\
\texttt{:enabled} A boolean controlling whether the pane is enabled. \\
\texttt{:background} The background color of the pane. \\
\texttt{:foreground} The foreground color of the pane. \\
\texttt{:font} The default font for the pane.
:horizontal-scroll
t, :without-bar, or nil. If true the pane can scroll horizontally.

:vertical-scroll
t, :without-bar, or nil. If true the pane can scroll vertically.

:visible-border
A boolean or a keyword controlling whether the pane has a border, for some pane classes.

:internal-border
A non-negative integer, or nil. Controls the width of the internal border.

:cursor
A keyword naming a built-in cursor, or a cursor object, or nil.

:pane-menu
Specifies a menu to be raised by the :post-menu gesture.

:drop-callback
Specifies a drop callback for output-pane or interface (and on Cocoa, list-panel or tree-view).

:drag-callback
Specifies a drag callback for list-panel or tree-view.

:scroll-if-not-visible-p
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.

:toolbar-title A string.
<table>
<thead>
<tr>
<th>Accessors</th>
<th>simple-pane-enabled</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>simple-pane-background</td>
</tr>
<tr>
<td></td>
<td>simple-pane-foreground</td>
</tr>
<tr>
<td></td>
<td>simple-pane-font</td>
</tr>
<tr>
<td></td>
<td>simple-pane-cursor</td>
</tr>
<tr>
<td></td>
<td>simple-pane-scroll-callback</td>
</tr>
<tr>
<td></td>
<td>simple-pane-drop-callback</td>
</tr>
<tr>
<td></td>
<td>simple-pane-drag-callback</td>
</tr>
<tr>
<td>Readers</td>
<td>simple-pane-horizontal-scroll</td>
</tr>
<tr>
<td></td>
<td>simple-pane-vertical-scroll</td>
</tr>
<tr>
<td></td>
<td>simple-pane-visible-border</td>
</tr>
</tbody>
</table>

**Description**

*enabled* determines whether the pane is enabled. The default value is `t`. Note that changing the enabled state of a visible pane changes its appearance.

*background* and *foreground* are colors specified using the Graphics Ports color system. Additionally on Cocoa, the special value `:transparent` is supported, which makes the pane’s background match that of its parent.

*font* should be font, a font description, or `nil`.

The value for *visible-border* can be any of the following, with the stated meanings where applicable:

- `nil` Has no border.
- `t` Has a border.
- `:default` Use the default for the window type.
- `:outline` Add an outline border.

There are various platform/pane class combinations which do not respond to all values of *visible-border*. For instance, on Windows XP with the default theme, *text-input-choice* and *option-pane* always have a visible border regardless of the value of *visible-border*, while other classes including *display-pane*, *text-input-pane*, *list-panel*, *editor-pane* and *graph-pane* have three distinct border styles, with *visible-border* `:default` meaning the same as *visible-border* `t`. 
If *internal-border* is non-nil, it should be a non-negative integer specifying the width of an empty region around the edge of the pane.

Any simple pane can be made scrollable by specifying \texttt{t} to \texttt{:horizontal-scroll} or \texttt{:vertical-scroll}. By default these values are \texttt{nil}, but some subclasses of \texttt{simple-pane} default them to \texttt{t} where appropriate (for instance \texttt{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 \texttt{:without-bar} for \texttt{:horizontal-scroll} or \texttt{:vertical-scroll}. See the example in \texttt{output-panes/scrolling-without-bar.lisp}.

The height and width of a scrollable simple pane can be specified by the initargs \texttt{:scroll-height} and \texttt{:scroll-width}, which have the same meaning as \texttt{:internal-min-height} and \texttt{:internal-min-width}. See the \textit{LispWorks CAPI User Guide} for more information about height and width initargs.

*cursor* specifies a cursor for the pane. \texttt{nil} means use the default cursor, and this is the default value. *cursor* can also be a cursor object as returned by \texttt{load-cursor}. The other allowed values are keywords naming built-in cursors which are supported on each platform as shown in the table below.

<table>
<thead>
<tr>
<th><em>cursor</em></th>
<th>Cocoa</th>
<th>Windows</th>
<th>Motif</th>
</tr>
</thead>
<tbody>
<tr>
<td>:busy</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>:i-beam</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>:top-left-arrow</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>:h-double-arrow</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>:v-double-arrow</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>:left-side</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>:right-side</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

\textbf{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

cursor | Cocoa | Windows | Motif  
--------|-------|---------|-------
:top-side | Yes   | Yes     | Yes   
:bottom-side | Yes   | Yes     | Yes   
:wait     | No     | Yes     | Yes   
:crosshair | Yes   | Yes     | Yes   
:gc-notification | No   | Yes     | Yes   
:top-left-corner | No   | Yes     | Yes   
:top-right-corner | No   | Yes     | Yes   
:bottom-left-corner | No   | Yes     | Yes   
:bottom-right-corner | No   | Yes     | Yes   
:hand     | Yes   | Yes     | Yes   
:fleur    | Yes   | Yes     | Yes   
:move     | Yes   | Yes     | Yes   
:closed-hand | Yes  | No      | No    
:open-hand | Yes   | No      | No    
:disappearing-item | Yes  | No      | No    

Table 1.2
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 \texttt{drop-object} using \texttt{drop-object-provides-format} and \texttt{drop-object-allows-drop-effect-p} to discover what the user is dragging. For \texttt{output-pane}, it can use \texttt{drop-object-pane-x} and \texttt{drop-object-pane-y} to query the mouse position relative to the pane. For \texttt{list-panel} and \texttt{tree-view}, it can use \texttt{drop-object-collection-index} or \texttt{drop-object-collection-item} to query where the user is attempting to drop the object and can call their \texttt{setf} functions to adjust this position. It should call \texttt{(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 \texttt{output-pane} and \texttt{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 \texttt{drop-object} as for the \texttt{:drag} stage, but can also obtain the object itself using \texttt{drop-object-get-object} for one of the formats in the list returned by \texttt{drop-object-provides-format}. Once the object is received, it should call \texttt{(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:

\[
drag-callback \text{ pane info } \Rightarrow \text{ plist}
\]

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-selec-tion). 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

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

Summary Gets the visible size of a pane.

Package capi
**simple-pane-visible-size**

**Signature**

```
simple-pane-visible-size pane => width, height
```

**Arguments**

`pane`  
A simple pane.

**Values**

`width`  
The width of the visible part of `pane`, or `nil`.

`height`  
The height of the visible part of `pane`, or `nil`.

**Description**

The generic function `simple-pane-visible-size` returns the size in pixels of the visible part of `pane`, that is the width and height of the viewport, not including any borders or scroll bars. If `pane` is not displayed the return values are `nil`.

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-width`

`with-geometry`

---

**simple-pane-visible-width**

**Generic Function**

**Summary**

Gets the visible width of a pane.

**Package**

`capi`

**Signature**

```
simple-pane-visible-width pane => result
```

**Arguments**

`pane`  
A simple pane.

**Values**

`result`  
The width of the visible part of `pane`, or `nil`.

**Description**

The generic function `simple-pane-visible-width` returns the width in pixels of the visible part of `pane`, that is the width 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-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

```lisp
(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))
    :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`. 

509
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:
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 `sorted-object`.

**Package**

capi

**Signature**

`sort-object-items-by sorted-object items => result`

**Arguments**

- `sorted-object`  An instance of `sorted-object` or a subclass.
- `items`  A list.

**Values**

- `result`  A permutation of `items`.

**Description**

The function `sort-object-items-by` sorts `items` according to the current sort type of `sorted-object`, as set by `sorted-object-sort-by`.

Note: if the sort type is reversed, `items` will be sorted in reverse order.

**See also**

- `sorted-object`
- `sorted-object-sort-by`
sorted-object

Class

Summary
Defines sorting operations.

Package
capi

Superclasses
standard-object

Subclasses
list-panel

Initargs
:sort-descriptions
A list.

Description
The sorted-object class defines sorting operations.

Each element of sort-descriptions is a sort description object, as returned by make-sorting-description. These define various sorting options and are used by sorted-object-sort-by and sort-object-items-by.

See also
make-sorting-description
sort-object-items-by
sorted-object-sort-by

sorted-object-sort-by

Generic Function

Summary
Sets the sorting type of a sorted-object.

Package
capi

Signature
sorted-object-sort-by pane new-sort-type &key allow-reverse

Arguments
pane An instance of sorted-object or a subclass.
new-sort-type The sort type to set.
allow-reverse A boolean.
The generic function `sorted-object-sort-by` sets the sort type of `pane` to `new-sort-type`. `new-sort-type` must match the type of one of the sort descriptions of `pane`.

If `allow-reverse` is non-nil and the sort type already matches `new-sort-type`, then the sort reverses the order of the `items`. The default value of `allow-reverse` is `t`.

If `pane` is a `list-panel`, then `sorted-object-sort-by` also calls `sort-object-items-by` to sort the items with the new sort type. For your own subclasses of `sorted-object` which are not subclasses of `list-panel`, if you need this behavior define an `:after` method that calls `sort-object-items-by`. You can also define `:after` methods on subclasses of `list-panel` to perform other tasks each time the items are sorted.

### start-gc-monitor

<table>
<thead>
<tr>
<th>Summary</th>
<th>Starts a Lisp Monitor window.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Package</td>
<td><code>capi</code></td>
</tr>
<tr>
<td>Signature</td>
<td><code>start-gc-monitor screen =&gt; result</code></td>
</tr>
<tr>
<td>Arguments</td>
<td><code>screen</code> A screen.</td>
</tr>
<tr>
<td>Values</td>
<td><code>result</code> A boolean.</td>
</tr>
<tr>
<td>Description</td>
<td>The function <code>start-gc-monitor</code> starts a Lisp Monitor window (otherwise known as the GC or Garbage Collector monitor) on the screen <code>screen</code>.</td>
</tr>
</tbody>
</table>
result is \texttt{t} if it started a Lisp monitor, and \texttt{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, \texttt{start-gc-monitor} is called automatically when the LispWorks IDE starts, but you can call \texttt{stop-gc-monitor} and \texttt{start-gc-monitor} any time.

See also \texttt{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.

```lisp
(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.
switchable-layout-visible-child
switchable-layout-combine-child-constraints

The switchable-layout has a description which is its list of children. The argument visible-child specifies the initially visible child (which defaults to the first of the children).

switchable-layout inherits most of its layout behavior from simple-layout as it only ever lays out one child at a time.

combine-child-constraints influences the initial size of the layout. When combine-child-constraints is nil 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 combine-child-constraints is non-nil, 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 combine-child-constraints is nil.

Example

```
(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)
```

There is a further example in the file examples/capi/layouts/switchable.lisp.
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
choice
layout

Initargs
: description The main layout description.
: items Specifies the tabs of the tab layout.
: visible-child-function Returns the visible child for a given selection in switchable mode.
: combine-child-constraints A generalized boolean which influences the initial size of the layout.
: key-function Specifies a function to use in referring to items in the items list.
: print-function The function used to print a name on each tab.
: callback-type The type of data passed to the callback function in callback mode.
: selection-callback The function called when a tab is selected, in callback mode.

Accessors tab-layout-visible-child-function

Readers tab-layout-combine-child-constraints

Description
A tab-layout has one of two distinct modes. It is in switchable mode if visible-child-function is supplied and non-nil. It is in callback mode otherwise.

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
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-panel` 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.

```lisp
(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 **tab-layout**.

**Package**

capi

**Signature**

tab-layout-panes tab-layout => panes

**Arguments**

- **tab-layout**
  
  A **tab-layout**.
Values  

panes  

A list.

Description  
The function `tab-layout-panes` returns the panes in a tab-layout. Note that this is not necessarily the same as the items of `tab-layout`, since `visible-child-function` and/or `key` may be specified.

See also  
tab-layout

tab-layout-visible-child  

Function

Summary  
Returns the visible child in a tab-layout.

Package  
capi

Signature  
`tab-layout-visible-child tab-layout => result`

Arguments  
`tab-layout`  
A tab-layout.

Values  
`result`  
A pane.

Description  
The function `tab-layout-visible-child` returns the currently-visible pane in a tab-layout.

See also  
tab-layout

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  
capi
### text-input-pane

**Class**

**Summary**

The class text-input-pane is a pane for entering a single line of text.

**Package**

capi

---

**Superclasses**

<table>
<thead>
<tr>
<th>choice</th>
</tr>
</thead>
<tbody>
<tr>
<td>text-input-pane</td>
</tr>
</tbody>
</table>

**Initargs**

<table>
<thead>
<tr>
<th>:visible-items-count</th>
</tr>
</thead>
<tbody>
<tr>
<td>An integer specifying the maximum length of the drop-down list, or the symbol :default.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>:popup-callback</th>
</tr>
</thead>
<tbody>
<tr>
<td>A function called just before the drop-down list appears, or nil.</td>
</tr>
</tbody>
</table>

**Description**

The text-input-choice class behaves in the same way as a text-input-pane, but has additional functionality. The element inherits from choice, and the choice items are used as the items to display when the user clicks on the button.

The callback is called when the user presses the Return key.

The selection-callback is called when the user selects an item using the drop-down list.

**Notes**

The text-input-pane initarg value enabled :read-only is not supported for text-input-choice on Microsoft Windows.

**Examples**

See examples/capi/elements/text-input-choice.lisp.

**See also**

<table>
<thead>
<tr>
<th>choice</th>
</tr>
</thead>
<tbody>
<tr>
<td>text-input-pane</td>
</tr>
</tbody>
</table>
1 CAPI Reference Entries

Superclasses  
titled-object
  simple-pane

Subclasses  
multi-line-text-input-pane
  password-pane
  text-input-choice

Initargs  
:text  The text in the pane.
:caret-position  The position of the caret in the text (from 0).
:max-characters  The maximum number of characters allowed.
:enabled  Controls the enabled state of the pane.
:completion-function  A function called to complete the text.
:in-place-completion-function  A function designator.
:file-completion  t, nil or a pathname designator.
:in-place-filter  A boolean.
:directories-only  A boolean.
:ignore-file-suffices  A list of strings or the keyword :default.
:callback-type  The type of arguments to callback.
:callback  A function usually called when the user presses Return.
:change-callback-type  The type of arguments to callback.
:change-callback
   A function called when a change is made.

:confirm-change-function
   A function called to validate a change. Note:
   Implemented for Motif only, not Microsoft
   Windows or Mac OS X.

:navigation-callback
   A function called when certain keyboard
   gestures occur in the pane.

:editing-callback
   A function called when editing starts or
   stops.

:gesture-callbacks
   A list of pairs (gesture . callback).

:complete-do-action
   A boolean.

:text-change-callback
   A function designator.

:buttons
   A plist specifying buttons to add, or t or
   nil.

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
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:

\[
in-place-completion-function \text{ pane string } \Rightarrow \text{ completions, start, end}
\]

where \text{pane} is the text-input-pane itself and \text{string} is the string to complete. When in-place completion is invoked \text{in-place-completion-function} is called with \text{pane} and a string containing the text of pane to the left of the cursor.

\text{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, \text{start} and \text{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 \text{in-place-completion-function} needs some dynamic information, it can put it in a property of the pane (using \text{capi-object-property}).

Note: The initarg :file-completion overrides :in-place-completion-function.

Note: The in-place completion mechanism uses gesture-call-backs 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.

\text{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 \texttt{t} then the in-place window can have a filter. Note that the filter needs to requested by a user gesture. \texttt{Control+Return} is the default in-place filter gesture. The default value of in-place-filter is \texttt{t}.

directories-only takes effect only if file-completion is used. If directories-only is \texttt{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 \texttt{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 \texttt{editor::*ignorable-file-suffices*} (see \texttt{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 pos-
sible 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 Win-
dows 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 key-words :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.
:orientation
The value is either :horizontal or :vertical. orientation controls the orientation of the toolbar. This is useful for multi-line-text-input-pane. The default value is :horizontal.

:adjust
The value is :top, :center, :centre or :bottom. adjust controls how the buttons are adjusted vertically relative to the text input pane. This is useful for multi-line-text-input-pane. The default value is :center.

:position
The value is :top, :bottom, :left or :right. position determines whether the buttons appear above, below, left or right of the text input pane. If :position is not supplied, then the buttons appear to the right of the pane.

The value nil for buttons means there are no buttons - this is the default. When buttons is true the buttons appear or not according to their specified values or their default values.

All of the button plists (for :ok, :cancel, :help and so on) can contain the following keys and values in addition to those mentioned above:

:enabled
A value that controls whether the button is enabled. (See the reader text-input-pane-buttons-enabled).

:image
The image to use for the button. This should be either a pathname or string naming an image file to load, a symbol giving the id of an image registered with register-image-translation, an image object as returned by load-image or an external-image. The default image is one of the symbols ok-but-
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)
This example uses a plist value for the `buttons` key: `cancel` to specify that the Cancel button is initially disabled:

```lisp
(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 shows how to specify a Help button which displays a help message:

```lisp
(defvar *help-message* "A long help message.")
```

```lisp
(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 completion-function in a text-input-pane.

Package capi

Signature text-input-pane-complete-text pane => result

Arguments pane A text-input-pane.
Values

result            A string, or nil.

Description

The function text-input-pane-complete-text calls the completion-function of pane with the current text. If this call is successful, then the text of pane is set to the result, and text-input-pane-complete-text returns this result. Otherwise, result is nil.

Note: the completion-function may return a list of completion candidates, in which case text-input-pane-complete-text prompts the user to select one of the candidates.

See also

text-input-pane

text-input-pane-copy

Function

Summary

Copies the selected text in a text-input-pane to the clipboard.

Package       capi

Signature       text-input-pane-copy  text-input-pane

Arguments       text-input-pane An instance of text-input-pane or a subclass.

Description

The function text-input-pane-copy performs the clipboard copy operation on the selected text in text-input-pane. It does nothing if there is no selection.

See also

clipboard
text-input-pane
text-input-pane-selection
text-input-pane-cut
text-input-pane-delete
text-input-pane-paste
text-input-pane-cut  

**Function**

**Summary** Cuts the selected text in a text-input-pane to the clipboard.

**Package** capi

**Signature** text-input-pane-cut text-input-pane

**Arguments**
- text-input-pane: An instance of text-input-pane or a subclass.

**Description** The function text-input-pane-cut performs the clipboard cut operation on the selected text in text-input-pane. It does nothing if there is no selection.

**See also**
- clipboard
text-input-pane
text-input-pane-selection
text-input-pane-copy
text-input-pane-delete
text-input-pane-paste

---

text-input-pane-delete  

**Function**

**Summary** Deletes the selected text in a text-input-pane.

**Package** capi

**Signature** text-input-pane-delete text-input-pane

**Arguments**
- text-input-pane: An instance of text-input-pane or a subclass.

**Description** The function text-input-pane-delete deletes the selected text in text-input-pane. It does nothing if there is no selection.
See also      clipboard
             text-input-pane
             text-input-pane-selection
             text-input-pane-cut
             text-input-pane-copy
             text-input-pane-paste

text-input-pane-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 sub-
                                        class.
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 capi

Signature text-input-pane-selection pane => start, end

Arguments pane A text-input-pane.

Values start, end Non-negative integers.

Description The function text-input-pane-selection returns as multiple values the bounding indexes of the selection in pane. That is, start is the inclusive index of the first selected character, and end is one greater than the index of the last selected character.

If there is no selection, then both start and end are the caret position in pane.

See also set-text-input-pane-selection text-input-pane text-input-pane-selected-text text-input-pane-selection-p

---

text-input-pane-selection-p Function

Summary Returns true if there is selected text in a text-input-pane.

Package capi

Signature text-input-pane-selection-p pane => selectionp

Arguments pane A text-input-pane.

Values selectionp A boolean.

Description The function text-input-pane-selection-p returns true if there is a selected region in pane and nil otherwise.
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  capi

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
The class **text-input-range** provides numeric input of integers in a given range (some systems refer to this as 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

```lisp
(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*

This class provides a pane that displays a single line of text.

Package capi

Superclasses titled-object
simple-pane
Subclasses  

message-pane

Initargs  

:text  The text to appear in the title pane.

Accessors  

title-pane-text

Description  

The most common use of title panes is as a title decoration for a pane, and so the class titled-object is provided as a class that supports placing title panes around itself.

A title-pane with text "Title" is created automatically when a titled-object is created with title "Title".

By default, a title-pane is constrained so that it cannot resize (that is, the values of visible-max-width and visible-max-height are t). This can be overridden by passing :visible-max-width nil or :visible-max-height nil.

Example  

(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))

See also  

display-pane
text-input-pane
editor-pane

titled-menu-object  

Class

Summary  

The class titled-menu-object is a subclass of menu-object which supports titles, and it is used by menus, menu components and menu items.

Package  

capi
Superclasses  menu-object
Subclasses  menu
          menu-component
          menu-item
Initargs  :title  The title for the object.
          :title-function  A setup callback which returns the title for the object, and optionally a mnemonic for the title.
Accessors  menu-title
          menu-title-function
Description  The simplest way to give a title to a titled-menu-object is to just supply a title string, and this will then appear as the title of the object.

Alternatively, a title-function can be provided which will be called when the menu is about to appear and which should return the title to use. By default title-function is called on the interface of the titled-menu-object, but this argument can be changed by passing the menu-object initarg setup-callback-argument.

To specify a mnemonic in the title returned by title-function, make title-function return the mnemonic as a second value. This value is interpreted in the same way as the mnemonic argument for menu.

Example
(capi:contain (make-instance 'capi:menu-item
    :title "Press Me")

(capi:contain (make-instance 'capi:menu-item
    :title-function #'(lambda (item)
      (princ-to-string
        (random 5))))))
### titled-object

**Class**

**Summary**
The class `titled-object` is a mixin class which provides support for decorating a pane with a title (a piece of text positioned next to the pane) and with a message (a piece of text below the pane).

**Package**
capi

**Subclasses**
- interface
- layout
- title-pane
- display-pane
- text-input-pane
- toolbar
- button-panel
- list-panel
- option-pane
- progress-bar
- output-pane
- slider

**Initargs**
- :title A title string for the pane (or nil).
- :title-args Initargs to the title `make-instance`.
- :title-font The font used for the title.
- :title-position The position of the title.
- :title-adjust How to adjust the title relative to the pane.
- :title-gap The gap between the title and the pane.
- :message A message string for the pane (or nil).
- :mnemonic-title A string specifying the title and a mnemonic. Applies only to the subclasses specified below.
- :message-gap The gap between the message and the pane.
Accessors

titled-object-title

titled-object-title-font

titled-object-message

titled-object-message-font

Description

The titled pane makes its title decoration from a title-pane and the message decoration from a message-pane.

The text of the title-pane is passed via the titled-object initarg title and the text of the message-pane is passed via the titled-object initarg message.

The initargs and font for the title-pane are passed via the titled-object initargs title-args and title-font respectively.

title-gap specifies the size in pixels of the gap between the title and the pane. The default value of title-gap is 3.

For subclasses other than interface, the font used for the message can be found by titled-object-message-font and set by (setf titled-object-message-font).

message-gap specifies the size in pixels of the gap between the message and the pane. The default value of message-gap is 3.

The message is always placed below the pane, but the title’s position can be adjusted by specifying title-position which can be any of the following.

:left Place the title to the left of the pane.
:right Place the title to the right of the pane.
:top Place the title above the pane.
:bottom Place the title below the pane.
:frame Place the title in a frame (like a groupbox) around the pane.

The title-adjust slot is used to adjust the title so that it is left justified, right justified or centered. The value of title-adjust can be any of the values accepted by the function pane-adjusted-offset, which are :left, :right, :top, :bottom, :center and :centre.
Note: title-adjust cannot handle both x and y. It is designed for cases like this:

{(capi:contain '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))
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
**Summary**

This class provides a pane containing toolbar buttons and panes.

**Package**

capi

**Superclasses**

collection
simple-pane
titled-object
toolbar-object

**Initargs**

:dividerp If t, a divider line is drawn above the toolbar, to separate it from the menu bar. The default value is nil.

:images A list of images.

:callbacks A list of callback functions.

:tooltips A list of tooltip strings used on Microsoft Windows.

:button-width The width of the toolbar buttons.

:button-height The height of the toolbar buttons.

:stretch-text-p A generalized boolean.

:image-width The width of images in the toolbar.

:image-height The height of images in the toolbar.

:default-image-set An optional image-set object which can be used to specify images. See toolbar-button and image-set for more details.

:flatp A generalized boolean.

**Readers**

toolbar-flat-p
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

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


An image object, as returned by load-image.

An image locator object

This allows a single bitmap to be created which contains several button images side by side. See make-image-locator for more information. On Microsoft Windows, this also allows access to bitmaps stored as resources in a DLL.

An integer

This is a zero-based index into the default-image-set of the toolbar or toolbar component in which the toolbar button is used.

Each image should be of the correct size for the toolbar. By default, this is 16 pixels wide and 16 pixels high.

help-key is interpreted as described for element.

remapped, if non-nil, should match the name of a menu-item in the same interface as the button. Then, the action of pressing the button is remapped to selecting that menu-item and calling its callback. The default value of remapped is nil.
Toolbar buttons can be made with an associated dropdown menu by passing the :dropdown-menu or :dropdown-menu-function initargs.

If dropdown-menu is non-nil then it should be a menu object to display for the button.

If dropdown-menu-function is non-nil then it should be a function which will be called with no arguments and should return a menu object to display for the button.

dropdown-menu-kind can have the following values:

- :button There is a separate smaller button for the dropdown menu next to the main button.
- :only There is no main button, only the smaller button for the dropdown.
- :delayed There is only one button and the menu is displayed when the user holds the mouse down over the button for some short delay. If the user clicks on the button then the normal callback is called.

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:

```lisp
(defun do-redo (data interface)
  (declare (ignore data interface))
  (capi:display-message "Doing Redo")
)
```

A simple interface:

```lisp
(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:

```lisp
(capi:display (make-instance 'redo))
```

This last example illustrates the use of :selected-image.
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**
capi

**Superclasses**
None

**Subclasses**
toolbar
toolbar-button
toolbar-component

**Initargs**
:enabled If t, the toolbar object is enabled.
:enabled-function
A function determining the enabled state.

**Accessors**
simple-pane-enabled
toolbar-object-enabled-function

**Description**
Any toolbar object may be disabled, by setting its enabled slot to nil. Disabling a toolbar or toolbar component prevents the user from interacting with any buttons contained in it.

All toolbar objects may also have an enabled-function specified. This is called whenever update-toolbar is called. If it returns t, the toolbar object will be enabled; if it returns nil, the object will be disabled.

**See also**
toolbar
toolbar-button
toolbar-component
update-toolbar

**top-level-interface**

**Generic Function**

**Summary**
Returns the top level interface containing a specified pane.
<table>
<thead>
<tr>
<th>Package</th>
<th>capi</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signature</td>
<td>top-level-interface pane</td>
</tr>
<tr>
<td>Description</td>
<td>Returns the top level interface that contains <code>pane</code>.</td>
</tr>
<tr>
<td>See also</td>
<td>top-level-interface-p</td>
</tr>
<tr>
<td></td>
<td>interface element</td>
</tr>
</tbody>
</table>

### top-level-interface-display-state

#### Generic Function

<table>
<thead>
<tr>
<th>Summary</th>
<th>Returns a value which indicates how the top level interface is displayed.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Package</td>
<td>capi</td>
</tr>
<tr>
<td>Signature</td>
<td>top-level-interface-display-state interface</td>
</tr>
<tr>
<td>Arguments</td>
<td><code>interface</code> A top level interface or dialog window</td>
</tr>
<tr>
<td>Description</td>
<td>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 <code>interface</code>. The following values can be returned:</td>
</tr>
</tbody>
</table>

- **:normal** The window is visible and has its normal size.
- **:maximized** The window is visible and has been maximized.
- **:iconic** The window is visible as an icon.
- **:hidden** The window is not visible.
These values can also be passed as the :display-state initarg when making a top level interface.

In addition, the function (setf top-level-interface-display-state) can be used to change the state of a top level interface. The value can be set to one of the above, or to :restore if the current state is :iconic or :hidden. When set to :restore, the state will become :normal or :maximized depending on how the interface was visible in the past.

See also
top-level-interface-p
top-level-interface-geometry
set-top-level-interface-geometry
interface

top-level-interface-geometry

Generic Function

Summary
Returns the geometry of the top level interface.

Package
capi

Signature
top-level-interface-geometry interface

Description
The generic function top-level-interface-geometry returns the coordinates of the given interface in a form suitable for use as the :best-x, :best-y, :best-width and :best-height initargs to interface. The value of interface should be a top level interface.

Example
;;; Define and display an interface.
(capi:define-interface test ()
  ()
  (:panes (panel capi:list-panel)))
(setq int (capi:display (make-instance 'test)))
;; Now manually position the interface somewhere.
;; Find where the interface is.
(multiple-value-setq (tx ty twidth theight)
  (capi:top-level-interface-geometry int))
;; 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
capi

Signature
top-level-interface-geometry-key interface => key, product-name

Arguments
interface A top level interface.

Values
key A symbol.
product-name 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

```lisp
(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:

```lisp
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-stateinterface
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
<table>
<thead>
<tr>
<th><strong>tracking-pinboard-layout</strong></th>
<th><strong>Class</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Summary</strong></td>
<td>A pinboard with automatic highlighting.</td>
</tr>
<tr>
<td><strong>Package</strong></td>
<td>capi</td>
</tr>
<tr>
<td><strong>Superclasses</strong></td>
<td>pinboard-layout</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>The class <em>tracking-pinboard-layout</em> 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 <em>input-model</em>. Therefore, you may not specify :motion in the <em>input-model</em> of a <em>tracking-pinboard-layout</em>. See <em>output-pane</em> for a description of <em>input-model</em>.</td>
</tr>
</tbody>
</table>
Example

(defun 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))))
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 descendents when the status of an item is changed.

:checkbox-change-callback

A function called when the status of an item is changed interactively.

:checkbox-initial-status

Specifies the initial status of specific items.
Accessors

- `tree-view-roots`
- `tree-view-children-function`
- `tree-view-image-function`
- `tree-view-state-image-function`
- `tree-view-leaf-node-p-function`
- `tree-view-retain-expanded-nodes`
- `tree-view-expandp-function`
- `tree-view-action-callback-expand-p`
- `tree-view-right-click-extended-match`
- `tree-view-has-root-line`
- `tree-view-checkbox-next-map`
- `tree-view-checkbox-parent-function`
- `tree-view-checkbox-status`
- `tree-view-checkbox-child-function`
- `tree-view-checkbox-change-callback`
- `tree-view-checkbox-initial-status`

Readers

- `tree-view-checkbox-status`

Description

The tree view pane allows the user to select between items displayed in a hierarchical list. Although it is a choice, only single selection interaction is supported. 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 \texttt{load-image}. Currently this must be a bitmap file.

A symbol
   The symbol must have been previously registered by means of a call to \texttt{register-image-translation}.

An image object, as returned by \texttt{load-image}.

An image locator object
   This allowing a single bitmap to be created which contains several button images side by side. See \texttt{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 treeview’s image list. This is generally only useful if the image list is created explicitly. See \texttt{image-list} for more details.

The \texttt{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 \texttt{nil} to indicate that there is no state image. See also \texttt{checkbox-status}, which overrides the \texttt{state-image-function}.

If \texttt{image-lists} is specified, it should be a plist containing the following keywords as keys. The corresponding values should be \texttt{image-list} objects.

\textbf{:normal}
   Specifies an \texttt{image-list} object that contains the item images. The \texttt{image-function} should return a numeric index into this \texttt{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 item’s descendents may change if a `checkbox-child-function` was supplied.
- A callback given by `checkbox-callback-function` will be called, if this was supplied.

By default checkboxes have three statuses indicated by images: un-checked(0), grey-checked(1) and checked(2). If an item is checked or un-checked, then all its descendents have the same status. If an item is grey-checked, then its descendents have various statuses. When the status of an item changes, all the descendents 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`
1 CAPI Reference Entries

```lisp
(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 tree-view is visible.

Package capi

Signature (tree-view-ensure-visible tree-view item)

Arguments

(tree-view A tree view.

(item A displayed item of tree-view.

Description The function tree-view-ensure-visible ensures that an
tree-view is visible, scrolling the tree view if neces-
item in a tree view is visible, scrolling the tree view if neces-
sary.

Note that item must be an item that is displayed in tree-view.

See also tree-view

(tree-view-expanded-p) Generic Function

Summary Gets and sets the expanded state of an item in a tree-view.

Package capi

Signature (tree-view-expanded-p tree-view item)

Signature (setf tree-view-expanded-p) on tree-view item

Arguments

(tree-view A tree-view.
```
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 an 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
```
tree-view-update-an-item tree-view item in-parent
```

### Description
The generic function `tree-view-update-an-item` is a synonym for `tree-view-update-item`. 

**Note:** `tree-view-update-an-item` is deprecated. Please use `tree-view-update-item` instead.

### See also
- `tree-view`
- `tree-view-update-item`

---

### tree-view-update-item

**Generic Function**

#### Summary
Updates an item in a `tree-view`.

#### Package
```
capi
```

#### Signature
```
tree-view-update-item tree-view item in-parent
```

#### Arguments
- `tree-view` A `tree-view`.
- `item` An item.
- `in-parent` A boolean.

#### Description
The generic function `tree-view-update-item` updates the item `item` in `tree-view`. This includes recomputing the text, images and children of `item`. This is useful when the data in `tree-view` changes, but the entire tree does not need recomputing.

When `in-parent` is non-nil, `tree-view-update-item` updates the children of the parent of `item`. This is useful when `item` is actually removed from `tree-view`, causing the children of its parent to be re-positioned.

#### See also
- `tree-view`
**undefine-menu**

*Macro*

Package `capi`

Signature `undefine-menu function-name &rest args`

Description This function undefines a menu created with `define-menu`.

See also `define-menu` `menu`

**unhighlight-pinboard-object**

*Generic Function*

Summary Removes the highlighting from a `pinboard-object`.

Package `capi`

Signature `unhighlight-pinboard-object pinboard object &key redisplay`

Description This removes the highlighting from a pinboard object if necessary, and then if `redisplay` is non-nil it redisplays it. The default value of `redisplay` is `t`.

To highlight a pinboard object use `highlight-pinboard-object`.

See also `highlight-pinboard-object` `pinboard-object`

**uninstall-postscript-printer**

*Function*

Summary Uninstalls a Postscript printer definition.

Package `capi`
Signature

uninstall-postscript-printer name &key if-does-not-exist deletep

Arguments

name A string.
if-does-not-exist One of nil or :error.
deletep A boolean.

Description

Uninstalls a PostScript printer definition for the given device name.

This applies only on GTK+ and Motif.

if-does-not-exist controls what happens if the named printer does not exist. The default value is :error.
deletep, if true, causes the printer to be removed for subsequent sessions as well as the current session, by deleting the file on the disk. The default value of deletep is nil.

See also

install-postscript-printer

unmap-typeout

Function

Package capi

Signature unmap-typeout collector-pane

Description

This switches the collector-pane out from its switchable layout, and brings back the pane that was there before map-typeout was called.

See also

map-typeout
with-random-typeout
collector-pane
**update-all-interface-titles**  
*Function*

**Summary** Updates interface window titles.

**Package** capi

**Signature** update-all-interface-titles

**Description** The function `update-all-interface-titles` can be used to update all the interface window titles when needed. This is useful when `interface-extend-title` may return a new, different, value.

`update-all-interface-titles` calls `update-screen-interface-titles` on all the screens.

**See also**
- `interface-extend-title`
- `update-screen-interface-titles`

**update-interface-title**  
*Generic Function*

**Summary** Updates the title of an interface window.

**Package** capi

**Signature** update-interface-title interface

**Arguments**
- `interface` A CAPI interface.

**Description** The generic function `update-interface-title` updates the title of interface `interface`. This is useful when `interface-extend-title` may return a new, different, value.

You can specialize `update-interface-title` if needed.

To update all the interface titles, use `update-all-interface-titles` or `update-screen-interface-titles`. 
See also

- interface-extend-title
- update-all-interface-titles
- update-screen-interface-titles

**update-pinboard-object**

*Function*

**Package** capi

**Signature**

`update-pinboard-object object`

**Description**

This function checks the object's constraints, and adjusts the object's size as necessary. It then forces the layout to redisplay the object at its new size. Finally, it returns `t` 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.
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
The macro `with-busy-interface` switches the cursor of the interface containing `pane` to be the busy cursor, evaluates `body`, 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.

`cursor` specifies the cursor to use while `body` is running. The default value is `:busy`. For other allowed values, see `simple-pane`.

`delay` specifies a time in seconds before the cursor is switched, so if `body` runs in less than `delay` seconds, then the cursor is not switched at all. This is usually more useful behavior than switching the cursor immediately. The default value of `delay` is 0.5.

`with-busy-interface` must be called in the process of the interface containing `pane`.

`with-busy-interface` has no effect on Cocoa.

See also `simple-pane`

### with-dialog-results

**Macro**

**Summary**

Displays a dialog and executes a body when the dialog is dismissed.

**Package**

`capi`

**Signature**

`with-dialog-results (&rest results) dialog-form &body body => :continuation, nil`

**Arguments**

- `results` Variables.
- `dialog-form` A function call form.
- `body` Forms.
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 \texttt{popup-confirm} or \texttt{prompt-for-string}, which take a \texttt{:continuation} keyword. You can define your own such functions provided that they call one of the CAPI functions, passing the received \textit{continuation} argument.

\textbf{Examples} 

On Microsoft Windows, GTK+ and Motif, this displays a dialog, calls \texttt{record-label-in-database} when the user clicks OK and then returns. On Cocoa, this creates a sheet and returns; \texttt{record-label-in-database} will be called when the user clicks OK.

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

Here is an example with skeleton code for using \texttt{with-dialog-results}. Note that the dialog function (\texttt{choose-file} below) that is called by \texttt{with-dialog-results} must take a \texttt{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 \texttt{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-confirm

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
The **with-document-pages** evaluates **body** repeatedly, with **page-var** bound to the number of the page to print on each iteration. It is used by applications providing Page on Demand printing.

The **first-page** and **last-page** arguments are evaluated to yield the page numbers of the first and last pages in the document. **with-document-pages** takes care of **first-page** and **last-page** when the user sets them in **print-dialog**, by evaluating **body** for the pages that are in the intersection of what user chose and the other arguments.

See also

- **print-dialog**
- **with-page**
- **with-print-job**

### with-external-metafile

**Macro**

**Summary**

Creates a metafile on disk using Graphics Ports operations.

**Package**

capi

**Signature**

```lisp
(with-external-metafile (var &key pane bounds format pathname) &body body => nil)
```

**Arguments**

- **var**
  A variable.

- **pane**
  A graphics port, or **nil**.

- **bounds**
  A list of four integers. Can also be **nil** on Microsoft Windows.

- **format**
  One of **:enhanced** or **:windows**. Used only on Microsoft Windows.

- **pathname**
  A pathname or string.

- **body**
  Code containing Graphic Ports operations that draw to **var**.
**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:

-%x%  
The x position of the pane.

-%y%  
The y position of the pane.

-%object%  
The object whose geometry this is.

-%child%  
The same as %object% (kept for 3.1 compatibility).

-%ratio%  
Ratio information.

The following variables give the external size and external constraints (see the LispWorks CAPI User Guide for a description of width and height constraints):

-%width%  
The width in pixels of the pane.

-%height%  
The height in pixels of the pane.

-%min-width%  
The minimum width of the pane.

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

call-instance

element

goto-

code

goto-

line

goto-

code

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/meta-

file.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**

- **stream** A variable.
- **printer** A printer or `nil`.
- **tab-spacing** An integer.
- **interactive** A boolean.
- **jobname** 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 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".
with-page

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

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  
\textbf{wrap-text} text width &key start end => strings

Arguments 
- \textit{text}  A string.
- \textit{width}  A positive integer.
- \textit{start, end} Bounding index designators of text.

Values 
- \textit{strings} A list of strings.

Description 
The function \texttt{wrap-text} takes a string \textit{text} and returns a list of strings, each of which is no longer than \textit{width}. Together the strings in \texttt{strings} contain all the non-whitespace characters of \textit{text} between \textit{start} and \textit{end} and are suitable for displaying this text on multiple lines of length \textit{width}.

See also  
\texttt{wrap-text-for-pane}

\textbf{wrap-text-for-pane}  
Function

Summary 
Wraps text for a given pane.

Package  
capi

Signature  
\texttt{wrap-text-for-pane} pane text &key external-width visible-width font start end => strings

Arguments 
- \textit{text}  A string.
- \textit{pane}  A displayed CAPI pane.
- \textit{external-width}  An integer or \texttt{nil}.
- \textit{visible-width}  An integer or \texttt{nil}.
- \textit{font}  A font object.
- \textit{start}  An integer.
- \textit{end}  An integer or \texttt{nil}.
1 CAPI Reference Entries

Values

| strings | A list of strings. |

Description

The function `wrap-text-for-pane` takes a string `text` and returns a list of strings. Together the strings in `strings` contain all the non-whitespace characters of `text` and are suitable for displaying this text on `pane`. That is, each string has a display width no greater than the width of `pane` when drawn using the font of `pane`. The arguments `start` and `end` are used as bounding index designators for `text` and characters outside these bounds are ignored.

If `visible-width` is non-nil then text is wrapped to that width. Otherwise, if `external-width` 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 `visible-width` and `external-width` are `nil`, then the text is wrapped to the current visible width of the pane. The default value of both `visible-width` and `external-width` is `nil`.

The `font` is used to perform the wrapping calculations. If it is `nil` (the default), then the `graphics-state-font` is used for panes such as `output-pane` that have a `graphics-state` and the `simple-pane-font` is used for other panes.

See also

`wrap-text`

x-y-adjustable-layout

Class

Summary

The class `x-y-adjustable-layout` provides functionality for positioning panes in a space larger than themselves (for example, it is used to choose whether to center them, or left justify them).

Package
capi

Superclasses
layout
Subclasses

simple-layout
grid-layout

Initargs

:x-adjust The adjust value for the \textit{x} direction.
:y-adjust The adjust value for the \textit{y} direction.

Accessors

layout-x-adjust
layout-y-adjust

Description

The values \textit{x-adjust} and \textit{y-adjust} 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 \texttt{(keyword n)} where \textit{n} is an integer. These values of \textit{adjust} are interpreted as by \texttt{pane-adjusted-position}.

\texttt{:top} is the default for \textit{y-adjust} and \texttt{:left} is the default for \textit{x-adjust}.

Example

Note: \texttt{column-layout} is a subclass of \texttt{x-y-adjustable-layout}.

```lisp
(setq column (capi:contain
  (make-instance
    'capi:column-layout
    :description (list
      (make-instance
        'capi:push-button
        :text "Ok")
      (make-instance
        'capi:list-panel
        :items '(1 2 3 4 5)))))

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

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

See also

pane-adjusted-position
1 CAPI 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**

**graphics-ports**

**Signature**

```
analyze-external-image external-image => width height color-table number
```

**Arguments**

- `external-image` An external-image.

**Values**

- `width` An integer.
- `height` An integer.
- `color-table` 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

*transform* A transform.

*sx* A real number.

*sy* A real number.

Description

The **apply-scale** function modifies *transform* such that a scaling of *sx* in *x* and *sy* in *y* 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

`graphics-ports`

Signature

`apply-translation transform dx dy`

Arguments

*transform* A transform.

*dx* A real number.

*dy* A real number.

Description

The **apply-translation** function modifies *transform* such that a translation of *(dx dy)* 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

`graphics-ports`
augment-font-description  

**Arguments**
- `fdesc` A font description.
- `font-attribute` A font attribute.

**Values**
- `return` A font description.

**Description**
The `augment-font-description` function returns a font description that contains all the attributes of `fdesc` combined with the extra `font-attributes`. The `:stock` attribute is handled specially: it is omitted from `return`, unless it is the only attribute specified.

If an attribute appears in both `fdesc` and a `font-attribute`, the value in the `font-attribute` is used. The contents of `fdesc` are not modified.

**See also**
- `make-font-description`

---

**clear-external-image-conversions**

**Summary**
Clears external image conversions for a port.

**Package**
`graphics-ports`

**Signature**
clear-external-image-conversions  

**Arguments**
- `external-image` An external image.
- `gp-or-null` A graphics port or `nil`.
- `free-image` A boolean.
- `all` A boolean.
- `errorp` A boolean.
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**
graphics-ports

**Signature**
clear-rectangle port x y width height

**Arguments**
- port: A graphics port.
- x: A real number.
- y: A real number.
- width: A real number.
- height: A real number.

**Description**
The `clear-rectangle` function draws the rectangle specified by x, y, width, and height in port’s background color. All other graphics state parameters are ignored.

compress-external-image

**Function**

**Summary**
Compresses DIB data in an external image.

**Package**
graphics-ports

**Signature**
compress-external-image external-image => result

**Arguments**
- external-image: An external-image.

**Values**
- result: The difference in bytes between size of the original image and the size of the compressed version.

**Description**
The `compress-external-image` function converts the external-image 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 $x$ coordinates of the end of each of the characters in a string if the string was printed to a graphics port.

**Package**

graphics-ports

**Signature**

`compute-char-extents port string &optional font => extents`

**Arguments**

- `port` A CAPI pane.
- `string` A string.
- `font` A font.

**Values**

`extents` An array of integers.

**Description**

Returns the *extents* of the characters in `string` in the font associated with `port`, or the `font` given. The extents are an array, one element per character, which gives the ending $x$ coordinate of that character if the string was drawn to `port`.

*Note:* To compute the extents of the entire string for a given `port` or `font`, use `port-string-width` or `get-string-extent`.

**See also**

- `get-string-extent`
- `port-string-width`

**convert-external-image**

*Function*

**Summary**

Returns an image derived from an external image format.

**Package**

graphics-ports
convert-external-image

Signature: convert-external-image gp external-image
            &key cache force-new => image

Arguments:
- gp: A CAPI pane.
- external-image: An external image.
- cache: A boolean.
- force-new: A boolean.

Values:
- image: An image.

Description:
The **convert-external-image** function returns an image derived from external-image. The image is ready for drawing to the given graphics port.

If cache is non-nil image conversions are cached in the external-image. The default value of cache is nil.

If force-new is non-nil a new image is always created, and put in the cache. The default value of force-new is nil.

---

convert-to-font-description

Function

Summary:
Converts a font-spec to a font description.

Package: graphics-ports

Signature: convert-to-font-description port font-spec => fdesc

Arguments:
- port: A graphics port
- font-spec: A font description object, font or symbol

Values:
- fdesc: A font-description

Description:
The function **convert-to-font-description** converts font-spec to a font description object fdesc for the graphics port port. If font-spec is a font, then its description is returned. If
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  
\textbf{copy-pixels} to-port from-port to-x to-y width height from-x from-y &\texttt{rest} \ \texttt{args}

Arguments  
to-port \hspace{2em} \text{A graphics port.}  
from-port \hspace{2em} \text{A graphics port.}  
to-x \hspace{2em} \text{A real number.}  
to-y \hspace{2em} \text{A real number.}  
width \hspace{2em} \text{A real number.}  
height \hspace{2em} \text{A real number.}  
from-x \hspace{2em} \text{A real number.}  
from-y \hspace{2em} \text{A real number.}  

Description  
The \texttt{copy-pixels} function copies a rectangular area from one port to another. The \texttt{transform}, \texttt{mask}, \texttt{mask-x} and \texttt{mask-y} from the \texttt{to-port}'s graphics state are used. The \texttt{(to-x to-y)} is transformed according to \texttt{to-port}'s transform, but the image is not scaled or rotated. The \texttt{to-port} and \texttt{from-port} need not be the same depth and can be the same object. The \texttt{from-x} and \texttt{from-y} values are interpreted as pixel positions in the window coordinates of \texttt{from-port}, that is, they are not transformed by \texttt{from-port}'s transform.

\textbf{copy-transform}  
\textit{Function (inline)}

Summary  
Returns a copy of a transform.

Package  
\texttt{graphics-ports}

Signature  
\textbf{copy-transform} transform => result

Arguments  
transform \hspace{2em} \text{A transform.}

Values  
result \hspace{2em} \text{A transform.}
Description  The \texttt{copy-transform} function returns a copy of \texttt{transform}.

\textbf{create-pixmap-port}  

\textit{Function}

\textbf{Summary}  Creates a pixmap port and its window system representation.

\textbf{Package}  \texttt{graphics-ports}

\textbf{Signature}  
\texttt{create-pixmap-port pane width height &key background collect relative clear => pixmap-port}

\textbf{Arguments}  
\begin{itemize}
  \item \textit{pane}  A graphics port for a window.
  \item \textit{width}  An integer.
  \item \textit{height}  An integer.
  \item \textit{background}  A color designator.
  \item \textit{collect}  A boolean.
  \item \textit{relative}  A boolean.
  \item \textit{clear}  A list or \texttt{t}.
\end{itemize}

\textbf{Values}  
\begin{itemize}
  \item \texttt{pixmap-port}  A pixmap graphics port.
\end{itemize}

\textbf{Description}  
The \texttt{create-pixmap-port} function creates a pixmap-port and its window system representation. The \textit{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 \textit{background} is used to initialize the \texttt{graphics-state-background}.

If \textit{clear} is \texttt{t}, the pixmap is cleared to its background color, otherwise the initial pixel values will be non-deterministic. If \textit{clear} is a list of the form \((x y width height)\), only that part of the pixmap is cleared initially. The default value is \texttt{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**  
`graphics-ports`

**Description**  
The `*default-image-translation-table*` 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**  
`load-image`

**define-font-alias**  
**Function**

**Summary**  
Defines an alias for a font.

**Package**  
`graphics-ports`
define-font-alias

Signature
define-font-alias  keyword  font

Arguments
keyword  A keyword.
font  A font.

Description
The function define-font-alias defines keyword as an alias for font.

destroy-pixmap-port

Function
Summary
Destroys a pixmap port, thereby freeing any window system resources it used.

Package
graphics-ports

Signature
destroy-pixmap-port  pixmap-port

Arguments
 pixmap-port  A pixmap port.

Description
The destroy-pixmap-port function destroys a pixmap-port, freeing any window system resources.

dither-color-spec

Function
Summary
Returns t if the color specification for a given pixel should result in a pixel that is on in a 1 bit dithered bitmap.

Package
graphics-ports

Signature
dither-color-spec  rgb-color-spec  y  x

Arguments
 rgb-color-spec  An RGB specification.
y  An integer.
x  An integer.
Values

result  A boolean.

Description

The dither-color-spec returns t if rgb-color-spec should result in a pixel that is on in a 1-bit dithered bitmap. The current set of dithers is used in the decision.

Note: dithers do not affect drawing or the antialiasing that occurs when drawing in Cocoa.

See also

initialize-dithers
make-dither
with-dither

draw-arc

Function

Summary

Draws an arc.

Package

graphics-ports

Signature

draw-arc port x y width height start-angle sweep-angle
&rest args &key filled

Arguments

port  A graphics port.

x  A real number.

y  A real number.

width  A real number.

height  A real number.

start-angle  A real number.

sweep-angle  A real number.

filled  A boolean.

args  General graphics port drawing arguments.
The draw-arc function draws an arc contained in the rectangle from \((x, y)\) to \((x+\text{width}, y+\text{width})\) 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

port A graphics port.
description A description sequence.
filled A boolean.
args 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 \(x\ y\ \text{width}\ \text{height}\ \text{start-angle}\ \text{sweep-angle}\). See draw-arc for more information.

See also draw-arc
**draw-character**  

*Function*

**Summary**  
Draws a character in a given graphics port.

**Package**  
*graphics-ports*

**Signature**  
`draw-character port character x y &rest args &key block`

**Arguments**
- `port`  
  A graphics port.
- `character`  
  A character.
- `x`  
  A real number.
- `y`  
  A real number.
- `block`  
  A boolean.
- `args`  
  General graphics port drawing arguments.

**Description**

The `draw-character` function draws the character at \((x, y)\) on the port. The `transform`, `foreground`, `background`, `operation`, `stipple`, `pattern`, `mask`, `mask-x`, `mask-y` and `font` from the port's graphics state (see `make-graphics-state`) are all used. The \((x, y)\) specifies the leftmost point of the character's baseline.  

`block`, if non-nil, causes the character to be drawn in a character cell filled with the port's graphics state background.  

**Note:** The Graphics State slot `operation` is not supported for drawing text on Windows.

**draw-circle**  

*Function*

**Summary**  
Draws a circle.

**Package**  
*graphics-ports*

**Signature**  
`draw-circle port x y radius &rest args &key filled`
The `draw-circle` function draws a circle of the given radius 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 circle is filled with the foreground color.

Example

\[
\begin{align*}
(gp:\text{draw-circle} \text{ port} \ 100 \ 100 \ 20) \\
(gp:\text{draw-circle} \text{ port} \ 100 \ 100 \ 50 \\
&\quad :\text{filled} \ t \\
&\quad :\text{foreground} \ :\text{green})
\end{align*}
\]

`draw-ellipse`  

Function

Summary

Draws an ellipse.

Package

`graphics-ports`

Signature

`draw-ellipse \ port \ x \ y \ x-radius \ y-radius \ &rest \ args \ &key \ filled`

Arguments

\begin{align*}
\text{port} & \quad \text{A graphics port.} \\
x & \quad \text{A real number.} \\
y & \quad \text{A real number.} \\
x-radius & \quad \text{A real number.}
\end{align*}
y-radius  A real number.

radius  A real number.

args  General graphics port drawing arguments.

filled  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-\(\text{nil}\), the ellipse is filled with the foreground color.

Example  

```lisp
(gp:draw-ellipse port 100 100 20 40)

(gp:draw-ellipse port 100 100 50 10
   :filled t
   :foreground :green)
```

**draw-image**  

Function

Summary  Displays an image on a graphics port at a given position.

Package  `graphics-ports`

Signature  

```lisp
draw-image port image to-x to-y &rest args &key from-x from-y to-width to-height from-width from-height global-alpha
```

Arguments  

- `port`  A graphics port.
- `image`  An image.
- `to-x`  A real number.
- `to-y`  A real number.
- `args`  General graphics port drawing arguments.
- `from-x`  A real number.
- `from-y`  A real number.
**draw-image**

**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.

**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.
from-x A real number.
from-y A real number.
to-x A real number.
to-y A real number.
args General graphics port drawing arguments.

Description
The draw-line function draws a line from (from-x from-y) to (to-x to-y). The graphics state parameters transform, foreground, background, operation, thickness, scale-thickness, dashed, dash, line-end-style and mask are used. Additionally on Unix only, stipple, pattern, mask-x, mask-y are used.

See also draw-lines

draw-lines

Function
Summary Draws several lines between pairs of two given points.
Package graphics-ports
Signature draw-lines port description &rest args
Arguments
port A graphics port.
description A description sequence.
args General graphics port drawing arguments.

Description
The draw-lines function draws several lines as specified by the description sequence. This is usually more efficient than making several calls to draw-line. The description argument is a sequence of values of the form x1 y1 x2 y2. See draw-line for more information.

See also draw-line
**draw-point**  
*Function*

**Summary**  
Draws a pixel at a given point.

**Package**  
graphics-ports

**Signature**  
\texttt{draw-point port x y &rest args}

**Arguments**  
\begin{itemize}
  \item \texttt{port}  
    A graphics port.
  \item \texttt{x}  
    A real number.
  \item \texttt{y}  
    real number.
  \item \texttt{args}  
    General graphics port drawing arguments.
\end{itemize}

**Description**  
The \texttt{draw-point} function draws a single-pixel point at \((x \; y)\). The \texttt{transform}, \texttt{foreground}, \texttt{background}, \texttt{operation} and \texttt{mask} slots of the graphics state are used. Additionally on Unix only, \texttt{stipple}, \texttt{pattern}, \texttt{mask-x}, \texttt{mask-y} are used.

**See also**  
draw-points  
set-graphics-state

---

**draw-points**  
*Function*

**Summary**  
Draws pixels at given points.

**Package**  
graphics-ports

**Signature**  
\texttt{draw-points port description &rest args}

**Arguments**  
\begin{itemize}
  \item \texttt{port}  
    A graphics port.
  \item \texttt{description}  
    A description sequence.
  \item \texttt{args}  
    General graphics port drawing arguments.
\end{itemize}
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**

**Summary**

Draws a polygon.

**Package**

`graphics-ports`

**Signature**

```
draw-polygon port points &rest args &key filled closed fill-rule
```

**Arguments**

- `port` A graphics port.
- `points` A description sequence.
- `filled` A boolean.
- `closed` A boolean.
- `fill-rule` A keyword.
- `args` 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.
See also draw-polygons

**draw-polygons**

*Function*

**Summary**
Draws several polygons.

**Package**
graphics-ports

**Signature**

draw-polygons port description &rest args
 &key filled closed fill-rule

**Arguments**
- **port** A graphics port.
- **description** A sequence of sequences of real numbers.
- **filled** A boolean.
- **closed** A boolean.
- **fill-rule** A keyword.
- **args** General graphics port drawing arguments.

**Description**
The **draw-polygons** function draws several polygons. The **description** argument should be a sequence containing sequences with alternating x and y values representing the vertices. The **description** arguments consists of groups of points as in **draw-polygon**. 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 polygons are 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.

**Example**
This draws two hexagons, one inside the other:
**Function**

**draw-rectangle**

**Summary**

Draws a rectangle.

**Package**

graphics-ports

**Signature**

\( \text{draw-rectangle port x y width height &rest args &key filled} \)

**Arguments**

- **port**
  A graphics port.
- **x**
  A real number.
- **y**
  A real number.
- **width**
  A real number.
- **height**
  A real number.
- **filled**
  A boolean.
- **args**
  General graphics port drawing arguments.

**Description**

The `draw-rectangle` function draws a rectangle whose corners are \((x, y), (x+width, y), (x+width, y+height)\) and \((x, y+height)\). The `filled` keyword, if non-nil, causes a filled rectangle to be drawn. While the exact results are host-specific, it is intended that a filled rectangle does not include the lines \((x = x+width)\) and \((y = y+height)\) while a non-filled rectangle does. This function works correctly if the `port`'s transform includes rotation. The graphics state parameters `transform`, `foreground`,
background, operation, thickness, scale-thickness, dashed, dash, line-joint-style and mask are used. Additionally on Unix only, stipple, pattern, mask-x, mask-y are used.

See also  
draw-rectangles

draw-rectangles

Function

Summary  
Draws several rectangles.

Package  
graphics-ports

Signature  
draw-rectangles port description &rest args &key filled

Arguments  
port  
A graphics port.

description  
A description sequence.

filled  
A boolean.

args  
General graphics port drawing arguments.

Description  
The draw-rectangles function draws several rectangles as specified in description which consists of a group of values given as x y width height. The filled keyword if non-nil causes filled rectangles to be drawn. While the exact results are host-specific, it is intended that a filled rectangle does not include the lines (x = x+width) and (y = y+height) while a non-filled rectangle does. This function works correctly if the port’s transform includes rotation. The graphics state parameters transform, foreground, background, operation, thickness, scale-thickness, dashed, dash, line-joint-style and mask are used. Additionally on Unix only, stipple, pattern, mask-x, mask-y are used.

See also  
draw-rectangle
**draw-string**  

**Function**

**Summary**  
Draws a string with the baseline positioned at a given point.

**Package**  
graphics-ports

**Signature**  
`draw-string port string x y &rest args &key start end block`

**Arguments**

- `port`  
A graphics port.

- `string`  
A string.

- `x`  
A real number.

- `y`  
A real number.

- `start`  
A real number.

- `end`  
A real number.

- `block`  
A boolean.

- `args`  
General graphics port drawing arguments.

**Description**

Draws the string with the baseline starting at `(x y)`. The transform, foreground, background, operation, stipple, pattern, mask, mask-x, mask-y and font from the port’s graphics state (see `make-graphics-state`) are all used. `start` and `end` specify which elements of the string to draw. `block`, if non-nil, causes each character to be drawn in a character cell filled with the port’s graphics state background.

By default, `start` is 0.

You can draw with the system highlight by setting Graphics State slots `foreground :color_highlighttext` and `background :color_highlight`.

**Note:** The Graphics State slot operation is not supported for drawing text on Windows.

**See also**  
make-graphics-state
**ensure-gdiplus**

*Function*

**Summary**

Ensures GDI+ is present and running, or shuts it down. Needed only when writing FLI graphics code on Windows.

**Package**

`graphics-ports`

**Signature**

`ensure-gdiplus &key event-func force shutdown => result`

**Arguments**

- `event-func` A function, or `nil`.
- `force` A boolean.
- `shutdown` A boolean.

**Values**

- `result` A boolean.

**Description**

The function `ensure-gdiplus` checks that the GDI+ module `gdiplus.dll` is loaded and that `GdiplusStartup` has been called, or shuts down GDI+.

Most users will not need to call `ensure-gdiplus`. This is because when LispWorks itself uses GDI+, for instance via `read-external-image`, it calls `ensure-gdiplus` automatically, and never shuts GDI+ down.

However, if your code uses GDI+ directly (by calling it through the Foreign Language Interface), then you should call `ensure-gdiplus` instead of using `GdiplusStartup` directly. Then, LispWorks will know that GDI+ has already started. This is the only circumstance in which you need to call `ensure-gdiplus`.

**Note:** `ensure-gdiplus` is implemented only in LispWorks for Windows.

If `shutdown` is `nil`, `ensure-gdiplus` ensures GDI+ is started, by the following steps:

1. Load the GDI+ module `gdiplus.dll`, if it is not already loaded.
2. If
   a) GDI+ was already started by a previous call to \texttt{ensure-gdiplus}, and
   b) \texttt{force} is \texttt{nil}, and
   c) \texttt{event-func} was either not passed or is \texttt{eq} to the value
      that was passed for point a)
   then \texttt{ensure-gdiplus} simply returns \texttt{nil}.

3. If GDI+ was already started, shut it down.

4. Start GDI+, and return the result of \texttt{GdiplusStartup}. This
   is 0 for success. For the meaning of other values, see the
   documentation of \texttt{gpStatus} in the MSDN.

   If \texttt{shutdown} is true, then if GDI+ was started \texttt{ensure-gdiplus}
   shuts it down, and returns \texttt{t}, otherwise \texttt{ensure-gdiplus}
   returns \texttt{nil}. The default value of \texttt{shutdown} is \texttt{nil}.

   The default value of both \texttt{event-func} and \texttt{force} is \texttt{nil}.

See also \texttt{read-external-image}

\textbf{external-image} \hspace{1cm} \textit{Class}

\textbf{Summary} \hspace{2cm} A class representing a color image.

\textbf{Package} \hspace{2cm} \texttt{graphics-ports}

\textbf{Description} \hspace{2cm} The class \texttt{external-image} provides a representation of a
color image that is subject to \texttt{write-external-image}, \texttt{read-external-image} and \texttt{convert-external-image} operations.

See also \texttt{convert-external-image}
\texttt{read-external-image}
\texttt{write-external-image}
**external-image-color-table**  
*Function*

Summary
Returns a vector containing RGB color specifications of an external image.

Package
`graphics-ports`

Signature
`external-image-color-table external-image => color-table`

Arguments
- `external-image` An external image.

Values
- `color-table` A color table.

Description
The `external-image-color-table` function returns a vector containing RGB color specifications representing the color table as specified in the external image. If the result is `nil`, 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
`graphics-ports`

Signature
`((setf external-image-color-table) replacement-color-table external-image)`

Arguments
- `external-image` An external image.
- `replacement-color-table` 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**  
*graphics-ports*

**Signature**  
`find-matching-fonts pane fdesc => fonts`

**Arguments**  
- `pane`  
  A CAPI pane.
- `fdesc`  
  A font description.

**Values**  
- `fonts`  
  A list of fonts.

**Description**  
The `find-matching-fonts` function returns a list of the font objects available for `pane` which match the attributes in `fdesc`. `nil` is returned if none match.

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-best-font`
- `list-all-font-names`
- `make-font-description`

---

**font-description**  
*Function*

**Summary**  
Returns a font description object for a given font.

**Package**  
*graphics-ports*

**Signature**  
`font-description font => fdesc`

**Arguments**  
- `font`  
  A font.

**Values**  
- `fdesc`  
  A font 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**

```
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**

```
fdesc font-attribute => value
```

**Arguments**
- `fdesc` A font description.
font-attribute  A font attribute.

Values

value  A font attribute value.

Description The \texttt{font-description-attribute-value} function returns the value of \texttt{font-attribute} in \texttt{fdesc}, or \texttt{:wild} if \texttt{font-attribute} is not specified in \texttt{fdesc}.

\textbf{font-fixed-width-p} \hspace{1cm} Function

Summary Returns \texttt{t} if a specified font is of a fixed width.

Package graphics-ports

Signature \texttt{font-fixed-width-p port &optional font \Rightarrow bool}

Arguments

\texttt{port}  A CAPI pane.

\texttt{font}  A font.

Values \texttt{bool}  A boolean.

Description The \texttt{font-fixed-width-p} function returns \texttt{t} if the font associated with \texttt{port}, or the optionally specified \texttt{font}, is fixed width.

\textbf{Note:} \texttt{editor-pane} supports variable width fonts on Microsoft Windows, GTK+ and Motif.

\textbf{free-image} \hspace{1cm} Function

Summary Frees the library resources allocated with an image.

Package graphics-ports

Signature \texttt{free-image port image}
Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>port</td>
<td>A CAPI pane.</td>
</tr>
<tr>
<td>image</td>
<td>An image.</td>
</tr>
</tbody>
</table>

Description
The free-image function frees the library resources associated with image. This should be done when an image is no longer needed.

**free-image-access**

*Function*

**Summary**
Frees an Image Access object.

**Package**
graphics-ports

**Signature**
free-image-access image-access

**Arguments**
image-access An Image Access object

**Description**
The function free-image-access discards image-access, which should be an Image Access object returned by make-image-access.

**See also**
image-access-transfer-from-image
image-access-transfer-to-image
image-access-pixel
make-image-access

**get-bounds**

*Function*

**Summary**
Returns the four values of the currently collected drawing extremes.

**Package**
graphics-ports

**Signature**
get-bounds pixmap-port => left, top, right, bottom
Arguments  

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pixmap-port</td>
<td>A graphics port.</td>
</tr>
</tbody>
</table>

Values  

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>left</td>
<td>An integer.</td>
</tr>
<tr>
<td>top</td>
<td>An integer.</td>
</tr>
<tr>
<td>right</td>
<td>An integer.</td>
</tr>
<tr>
<td>bottom</td>
<td>An integer.</td>
</tr>
</tbody>
</table>

Description  

The `get-bounds` function returns the four values `left`, `top`, `right`, `bottom` of the currently collected drawing extremes. The values can be used to get an image from the port.

Example  

```lisp
(with-pixmap-graphics-port (p1 pane width height :relative t)
  (with-graphics-rotation (p1 0.123)
    (draw-rectangle p1 100 100 200 120 :filled t :foreground :red)
    (get-bounds p1)))
```

produces the following output:

```
72
112
285
255
```

See also  

`make-image-from-port`

---

**get-character-extent**  

Function  

**Summary**  

Returns the extent of a character in pixels.

**Package**  

`graphics-ports`

**Signature**  

```
get-character-extent port character &optional font =>
  left, top, right, bottom
```

**Arguments**  

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>port</td>
<td>A CAPI pane.</td>
</tr>
</tbody>
</table>
### get-character-extent

**Summary**

The `get-character-extent` function returns the extent in pixels of the character in the font associated with `port`, or the font given.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>character</code></td>
<td>A character.</td>
</tr>
<tr>
<td><code>font</code></td>
<td>A font.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>left</code></td>
<td>An integer.</td>
</tr>
<tr>
<td><code>top</code></td>
<td>An integer.</td>
</tr>
<tr>
<td><code>right</code></td>
<td>An integer.</td>
</tr>
<tr>
<td><code>bottom</code></td>
<td>An integer.</td>
</tr>
</tbody>
</table>

### get-char-ascent

**Function**

Returns the ascent of a character in pixels.

**Package**

`graphics-ports`

**Signature**

```
get-char-ascent port character font => ascent
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>port</code></td>
<td>A CAPI pane.</td>
</tr>
<tr>
<td><code>character</code></td>
<td>A character.</td>
</tr>
<tr>
<td><code>font</code></td>
<td>A font.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ascent</code></td>
<td>An integer.</td>
</tr>
</tbody>
</table>

**Description**

The `get-char-ascent` function returns the ascent in pixels of the character in the font associated with `port`, or the font given.
get-char-descent

Summary
Returns the descent of a character in pixels.

Package
graphics-ports

Signature
get-char-descent port character font => descent

Arguments
port A CAPI pane.
character A character.
font A font.

Values
descent An integer.

Description
The get-char-descent function returns the descent in pixels of the character in the font associated with port, or the font given.

get-char-width

Summary
Returns the width of a character in pixels.

Package
graphics-ports

Signature
get-char-width port character font => width

Arguments
port A CAPI pane.
character A character.
font A font.

Values
width An integer.

Description
The get-char-width function returns the width in pixels of the character in the font associated with port, or the font given.
**get-enclosing-rectangle**

*Function*

**Summary**

Returns the smallest rectangle enclosing the given points.

**Package**

`graphics-ports`

**Signature**

`get-enclosing-rectangle &rest points => left, top, right, bottom`

**Arguments**

- `points` Real numbers.

**Values**

- `left` A real number.
- `top` A real number.
- `right` A real number.
- `bottom` A real number.

**Description**

The `get-enclosing-rectangle` function returns four values, describing the rectangle which exactly encloses the input points. The `points` argument must be a (possibly empty) list of alternating x and y values. If no `points` are given the function returns the null (unspecified) rectangle, which is four `nils`.

**get-font-ascent**

*Function*

**Summary**

Returns the ascent of a font.

**Package**

`graphics-ports`

**Signature**

`get-font-ascent port &optional font => ascent`

**Arguments**

- `port` A CAPI pane.
- `font` A font.

**Values**

- `ascent` An integer.
The get-font-ascent function returns the ascent in pixels of the font associated with port, or the font given.

**get-font-average-width**

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**

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.
get-font-descent function returns the descent in pixels of the font associated with port, or the font given.

**get-font-height**

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**

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  

Summary  Returns the graphics state object for a graphics port.

Package  graphics-ports

Signature  \texttt{get-graphics-state} \texttt{port} $\Rightarrow$ \texttt{state}

Arguments  \texttt{port}  A graphics port.

Values  \texttt{state}  A graphics state object.

Description  The \texttt{get-graphics-state} function returns the graphics state object for \texttt{port}. The individual slots can be accessed using the accessor functions.

See also  make-graphics-state

get-origin  

Summary  Returns the coordinate origin of a pixmap graphics port.

Package  graphics-ports

Signature  \texttt{get-origin} \texttt{pixmap-port} $\Rightarrow$ \texttt{x y}

Arguments  \texttt{pixmap-port}  A graphics port.

Values  \texttt{x}  An integer.
\texttt{y}  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 :relative \texttt{t}, 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

```lisp
(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.
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`
### image-access

**Arguments**
- `image-access`  
  An Image Access object
- `x`  
  An integer.
- `y`  
  An integer.

**Values**
- `color-rep`  
  A color reference.

**Description**
The function `image-access-pixel` returns the pixel value at position `x, y` in the Image Access object `image-access`.

The pixel value `color-rep` is a color representation like that returned by `convert-color`. If needed, `color-rep` can be converted to an RGB value using `unconvert-color`. `color-rep` can contain an alpha value, for images with an alpha channel.

The function `(setf image-access-pixel)` sets the value of the pixel at position `x, y` in the Image Access object `image-access`.

`image-access` must be an Image Access object returned by `make-image-access`.

**Example**
See these example files:
- `examples/capi/graphics/image-access.lisp`
- `examples/capi/graphics/image-access-alpha.lisp`

**See also**
- `image-access-pixels-from-bgra`
- `image-access-pixels-to-bgra`
- `image-access-transfer-to-image`
- `image-access-transfer-from-image`
- `free-image-access`
- `make-image-access`

---

**image-access-pixels-from-bgra**

**Function**

**Summary**
Copies a vector of pixel values into an Image Access object.
Package  graphics-ports

Signature  image-access-pixels-from-bgra  image-access  vector

Arguments  

  image-access  An Image Access object.

  vector  A vector.

Description  The function image-access-pixels-from-bgra copies all the pixels to the Image Access object image-access from the vector vector. vector should contain a sequence of integer values in the range 0-255 for blue, green, red and alpha of each pixel. This function is optimized for the case where vector has element type (unsigned-byte 8).

An error is signalled if vector is not of the correct length for the Image Access object, that is (* 4 width height) where width and height represent the size of image-access.

Note: image-access-pixels-to-bgra must be called after this function (similarly to (setf image-access-pixel)).

image-access must be an Image Access object returned by make-image-access.

Example  See the file examples/capi/graphics/image-access-bgra.lisp.

See also  image-access-pixel

image-access-pixels-to-bgra

image-access-pixels-to-bgra

Function

Summary  Copies pixel values from an Image Access object into a vector.

Package  graphics-ports
### Signature
image-access-pixels-to-bgra

### Arguments
- image-access: An Image Access object.
- vector: A vector.

### Description
The function `image-access-pixels-to-bgra` copies all the pixels in the Image Access object `image-access` into the vector `vector` as a sequence of integer values in the range 0-255 for the blue, green, red and alpha components of each pixel. This function is optimized for the case where `vector` has element type `(unsigned-byte 8)`.

An error is signalled if `vector` is not of the correct length for the Image Access object, that is `(\* 4 width height)` where `width` and `height` represent the size of `image-access`.

**Note:** `image-access-pixels-from-bgra` must be called before this function (similarly to `image-access-pixel`).

`image-access` must be an Image Access object returned by `make-image-access`.

### Example
See the file `examples/capi/graphics/image-access-bgra.lisp`.

### See also
- `image-access-pixel`
- `image-access-pixels-from-bgra`

---

### image-access-transfer-from-image

#### Function

#### Summary
Gets the pixel values from an image.

#### Package
graphics-ports

#### Signature
image-access-transfer-from-image

#### Arguments
- image-access: An Image Access object
The function `image-access-transfer-from-image` gets the pixel values from an `image` object, making them accessible via a corresponding Image Access object `image-access`.

`image-access` must be an Image Access object returned by `make-image-access`.

Notionally `image-access-transfer-from-image` transfers the pixel data from the window system into `image-access`, though it might do nothing on platforms where the window system allows direct access to the pixel data.

The pixel data can be accessed using `image-access-pixel`.

Example

See the file `examples/capi/graphics/image-access.lisp`.

See also

`image-access-transfer-to-image`

`image-access-pixel`

`free-image-access`

`make-image-access`

---

**image-access-transfer-to-image**

**Function**

**Summary**

Sets the pixel values in an `image`.

**Package**

`graphics-ports`

**Signature**

`image-access-transfer-to-image image-access`

**Arguments**

`image-access`  
An Image Access object

**Description**

The function `image-access-transfer-to-image` sets the pixel values in an `image` object from the values in a corresponding Image Access object `image-access`.

`image-access` must be an Image Access object returned by `make-image-access`. 
Notionally `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

- image-id: An image identifier.

Values

- loader: An image load function.

Description

The image-loader function returns the image load function that would be called to load the image associated with image-id in image-translation-table. If the image-id is not registered with a load function, the default image load function is returned. The default value of image-translation-table is *default-image-translation-table*.

See also

register-image-load-function
register-image-translation

---

**image-translation**

*Function*

Summary

Returns the translation for an image registered in its image translation table.

Package

graphics-ports

Signature

image-translation image-id &key image-translation-table => translation

Arguments

- image-id: An image identifier.

Values

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

**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 ^ 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

**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

rectangle  A list of integers.
dx        An integer.
dy        An integer.
dx-right  An integer.
dy-bottom An integer.

Description

The `inset-rectangle` function moves the left, top, right and bottom elements of rectangle inwards towards the center by the distances dx, dy, dx-right and dy-bottom respectively.

By default, dx-right is dx, and dy-bottom is dy.

**inside-rectangle**

Function

Summary

Determines if a point lies inside a rectangle.

Package

`graphics-ports`

Signature

`inside-rectangle rectangle x y => result`

Arguments

rectangle  A list of integers.
x        An integer.
y        An integer.

Values

result  A boolean.

Description

The `inside-rectangle` function returns t if the point (x y) is inside rectangle. The rectangle is expected to be ordered; if the rectangle is specified by (left right top bottom), then left must be less than right, and bottom must be less than top. The lines y = bottom and x = right are not considered to be inside the rectangle.
invalidate-rectangle

**Function**

**Summary**
Invalidate the rectangle associated with the object, which causes it to be redisplayed.

**Package**
graphics-ports

**Signature**
\[
\text{invalidate-rectangle object \&optional x y width height} => \text{result}
\]

**Arguments**
- \text{object} An instance of a subclass of graphics-ports-mixin or a subclass of pinboard-object.
- \text{x} A real number.
- \text{y} A real number.
- \text{width} A real number.
- \text{height} A real number.

**Values**
- \text{result} A boolean.

**Description**
By default it invalidates the whole rectangle, but this can be limited by passing the \&optional arguments.

**See also**
validate-rectangle

invert-transform

**Function**

**Summary**
Constructs the inverse of a transform.

**Package**
graphics-ports

**Signature**
\[
\text{invert-transform transform \&optional into} => \text{inverse}
\]

**Arguments**
- \text{transform} A transform.
A transform or nil.

Values inverse A transform.

Description This function constructs the inverse of transform. If $T$ is transform and $T'$ is its inverse, then $TT' = I$. If into is non-nil it is modified to contain $T'$ and returned, otherwise a new transform is constructed and returned.

**list-all-font-names**  
*Function*

Summary Finds the names of the available fonts.

Package `graphics-ports`

Signature `list-all-font-names pane => fdescs`

Arguments `pane` A graphics port.

Values `fdescs` A list of font description objects.

Description The function `list-all-font-names` returns a list of partially-specified font description objects which contain the "name" attributes for each known font that is available for `pane`.

On Microsoft Windows and Cocoa the "name" attributes are just the :family attribute.

On X11 the "name" attributes are :foundry and :family.

See also `font-description-attributes`
`find-matching-fonts`

**load-icon-image**  
*Function*

Summary Loads a Windows icon image, and returns the image object.
Package            graphics-ports

Signature          load-icon-image port id &key width height => image

Arguments          port         A graphics port or CAPI object.
                   id           A keyword, string or pathname.
                   width       The desired width in pixels, or nil.
                   height      The desired height in pixels, or nil.

Values             image        An image object.

Description        The load-icon-image function loads an icon specified by id which should be either a keyword describing a standard icon, or a string or a pathname naming a Windows format icon (.ico) file. In this case, the first icon in the file is loaded.
                   
                   The following keyword values of id are recognized:
                   
                   :sample       A rectangle
                   :hand         A cross in a circle
                   :ques         A question mark in a bubble
                   :bang         An exclamation mark in a triangle
                   :note         An 'I' in a bubble
                   :winlogo      The Windows logo
                   :warning      Same as :bang
                   :error        Same as :hand
                   :information  Same as :note

                   load-icon-image returns an image object which can be drawn to port using draw-image and which must be freed using free-image when no longer needed.
If width and height are specified, then the image is scaled accordingly. If width and height are nil then the dimensions are taken from the icon file. width defaults to nil and height defaults to width.

**Note:** `load-icon-image` is defined only in LispWorks for Windows.

See also
- `draw-image`
- `free-image`
- `load-image`

<table>
<thead>
<tr>
<th>load-image</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Summary</strong></td>
<td>Loads an image and returns the image object.</td>
</tr>
<tr>
<td><strong>Package</strong></td>
<td><code>graphics-ports</code></td>
</tr>
<tr>
<td><strong>Signature</strong></td>
<td><code>load-image gp id &amp;key cache type editable image-translation-table =&gt; image</code></td>
</tr>
<tr>
<td><strong>Arguments</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>Values</strong></td>
<td>image: An image object.</td>
</tr>
</tbody>
</table>
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.
The image is editable, but does not have an alpha channel.

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).

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

```lisp
(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`.

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**
Makes a dither matrix of a given size.

**Package**
graphics-ports

**Signature**
make-dither size => matrix

**Arguments**

\begin{itemize}
\item \textit{size} \hspace{1cm} An integer.
\end{itemize}

**Values**

\begin{itemize}
\item \textit{matrix} \hspace{1cm} A dither matrix.
\end{itemize}

**Description**
The \textit{make-dither} function makes a dither matrix of the given \textit{size}.

\textbf{Note}: dithers do not affect drawing or the antialiasing that occurs when drawing in Cocoa.

See also
dither-color-spec
initialize-dithers
with-dither

**make-font-description**

*Function*

**Summary**
Returns a new font description object containing given font attributes.

**Package**
graphics-ports
Signature: \texttt{make-font-description \&rest font-attribute* => fdesc}

Arguments:
- \texttt{font-attribute} \ A font attribute.

Values:
- \texttt{fdesc} \ A font description object.

Description:
The \texttt{make-font-description} function returns a new font description object containing the given font attributes. There is no error checking of the attributes at this point.

The \texttt{:stock} attribute is handled specially: it is omitted from \texttt{fdesc}, unless it is the only attribute specified.

See also:
- \texttt{augment-font-description}
- \texttt{convert-to-font-description}
- \texttt{find-best-font}
- \texttt{find-matching-fonts}
- \texttt{font-description}
- \texttt{merge-font-descriptions}

\textbf{make-graphics-state}

\textit{Function}

Summary: Creates a graphics state object.

Package: \texttt{graphics-ports}

Signature: \texttt{make-graphics-state \&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 => state}

Arguments:
- \texttt{transform} \ 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 any-
thing returned by the transform functions,
described in section "Graphics state trans-
forms" 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 draw-
ing 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 boole. The
default value is boole-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 trans-
formed 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 the 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.
---

**dashed**
A boolean, defaulting to `nil`. If `dashed` is `t` then lines are drawn as a dashed line using `dash` as the mark-space specifier.

**dash**
A list of two or more integer, or `nil`. 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 `dash` is `(4 4)`.

**line-end-style**
The value should be one of `:butt`, `:round` or `:projecting` and specifies how to draw the ends of lines. The default value is `:butt`.

**line-join-style**
The value should be one of `:bevel`, `:miter` or `:round` and specifies how to draw the areas where the edges of polygons meet. The default value is `:miter`.

**mask**
The value should be `nil` (the default) or a list of the form `(x y width height)`, defining a rectangle inside which the drawing is done. The mask is not tiled. A mask is not transformed by the `transform` parameter.

**mask-x**
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.

**mask-y**
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.

**font**
Either `nil` or a portable font description or font object to be used by the `draw-character` and `draw-string` functions. See "Portable font descriptions" in the *LispWorks CAPI User Guide* for details. The default value is `nil`.

---

668
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  
port  
A graphics port.

width  
A positive integer.

height  
A positive integer.

alpha  
A generalized boolean.

Values  
image  
An image object.

Description  
The function make-image makes a new blank, editable image object associated with port and of the given width and height. On Windows and Cocoa, if alpha is true, then the image will have an alpha channel.

The initial pixels in image are undefined. image is editable, that is, it is suitable for use with the Image Access API. To set the pixels, see make-image-access.

See also  
load-image  
make-image-access
**make-image-access**

**Generic Function**

**Summary**

Creates an Image Access object.

**Package**

`graphics-ports`

**Signature**

`make-image-access port image => image-access`

**Arguments**

- `port` A graphics port.
- `image` An `image` object.

**Values**

- `image-access` An Image Access object.

**Description**

The generic function `make-image-access` returns an Image Access object for the given `image` image.

`image` can be any `image` object returned by `make-image-from-port`. An `image` object returned by `load-image` is also suitable, but only if it is a Plain Image (see below).

`image-access` is used when reading and writing the pixel values of the image. For an overview of using Image Access objects, see the Graphics Ports chapter in the *LispWorks CAPI User Guide*.

**Note:** on some platforms (currently Windows) not every `image` object is a Plain Image. If needed, forcibly create a Plain Image suitable for passing to `make-image-access` as described in `load-image`.

**Note:** ensure that you eventually discard `image-access`, using `free-image-access`.

**Example**

See the file

`examples/capi/graphics/image-access.lisp`.

**See also**

- `free-image-access`
- `image-access-transfer-from-image`
- `image-access-transfer-to-image`
Function

**make-image-from-port**

Summary: Makes an image out of a specified rectangle of a graphics port’s contents.

Package: **graphics-ports**

Signature:

```
make-image-from-port port &optional x y width height => image
```

Arguments:
- **port**: A graphics port.
- **x**: An integer.
- **y**: An integer.
- **width**: An integer.
- **height**: An integer.

Values:
- **image**: An image.

Description: The **make-image-from-port** function makes an image out of the specified rectangle of the port’s contents. The default is the whole port, but a region can be specified using **x**, **y**, **width**, and **height**. The default value of **x** and **y** is 0.

Normally the image is freed automatically, when **port** 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.

See also: **externalize-image**
**make-sub-image**

**Function**

Summary  
Makes a new image from part of an image.

Package  
*graphics-ports*

Signature  
```
make-sub-image port image &optional x y width height  
=> sub-image
```

Arguments  
- `port`  
  A graphics port.
- `image`  
  An image.
- `x`  
  An integer.
- `y`  
  An integer.
- `width`  
  An integer.
- `height`  
  An integer.

Values  
- `sub-image`  
  An image.

Description  
The function **make-sub-image** makes a new image object from the rectangular region of the supplied image specified by `x`, `y`, `width` and `height`.

The default values of `x` and `y` are 0.

The default value of `width` is the width of `image`.

The default value of `height` is the height of `image`.

See also  
`image`

**make-transform**

**Function**

Summary  
Returns a new transform object initialized according to a set of optional arguments.

Package  
*graphics-ports*
Signature  
make-transform &optional a b c d e f => transform

Arguments  
a  A real number.
b  A real number.
c  A real number.
d  A real number.
e  A real number.
f  A real number.

Values  
transform  A transform.

Description  
The make-transform function returns a new transform object initialized according to the optional args. The default args make the unit transform. Default values are as follows: a and d are 1; b, c, e, and f are 0. The transform matrix is

\[
\begin{pmatrix}
a & b & 0 \\
c & d & 0 \\
e & f & 1 \\
\end{pmatrix}
\]

for generalized two dimensional points of the form \((x \ y \ 1)\).

merge-font-descriptions

Function

Summary  
Returns a font description containing the attributes of two specified font descriptions.

Package  
graphics-ports

Signature  
merge-font-descriptions fdesc1 fdesc2 => fdesc

Arguments  
fdesc1  A font description.
fdesc2  A font description.
Values  

\[ fdesc \]  
A font description.

Description  
The `merge-font-description` function returns a font description containing all the attributes of `fdesc1` and `fdesc2`. If an attribute appears in both `fdesc1` and `fdesc2`, the value in `fdesc1` is used. The `stock` attribute is handled specially: it is omitted from `fdesc`, unless it is the only attribute in `fdesc1` and `fdesc2`.

The contents of `fdesc1` and `fdesc2` are not modified.

See also  
`make-font-description`

---

### offset-rectangle

Function (inline)

**Summary**

Offsets a rectangle by a given distance.

**Package**

`graphics-ports`

**Signature**

`offset-rectangle rectangle dx dy`

**Arguments**

- `rectangle` A list of integers.
- `dx` A real number.
- `dy` A real number.

**Description**

The `offset-rectangle` function offsets the `rectangle` by the distance `(dx dy)`.

`rectangle` is a list `(left top right bottom)`.

---

### ordered-rectangle-union

Function

**Summary**

Returns the union of two rectangles.

**Package**

`graphics-ports`
Signature ordered-rectangle-union left-1 top-1 right-1 bottom-1
left-2 top-2 right-2 bottom-2
=> left, top, right, bottom

Arguments

left-1 A real number.
top-1 A real number.
right-1 A real number.
bottom-1 A real number.
left-2 A real number.
top-2 A real number.
right-2 A real number.
bottom-2 A real number.

Values

left A real number.
top A real number.
right A real number.
bottom A real number.

Description The ordered-rectangle-union function returns four values: the left, top, right and bottom of the union of the two rectangles specified in the arguments. The caller guarantees that each input rectangle is ordered, that is, the left values must be smaller or equal to the right values, and the top values must be greater than or equal to the bottom ones.

See also rectangle-union

pixblt

Summary Copies one area of a graphics port to another area of a different graphics port.
Package  

`graphics-ports`

Signature

`pixblt to-port operation from-port to-x to-y width height from-x from-y`

Arguments

- `to-port`: A graphics port.
- `operation`: A graphics state operation.
- `from-port`: A graphics port.
- `to-x`: A real number.
- `to-y`: A real number.
- `width`: A real number.
- `height`: A real number.
- `from-x`: A real number.
- `from-y`: A real number.

Description

The `pixblt` function copies one area of `from-port` to another area of `to-port` using the specified `operation` and `mask`. Both ports should be the same depth. The graphics port transforms are not used.

See the "Graphics state" section in the *LispWorks CAPI User Guide* for valid values for `operation`.

`pixmap-port`  

Class

Summary

The class of pixmap graphics port objects.

Package  

`graphics-ports`

Description

The `pixmap-port` class is the class of pixmap graphics port objects which can be used for drawing operations.
See also
create-pixmap-port
destroy-pixmap-port
with-pixmap-graphics-port

**port-height**

Function

Summary Returns the pixel height of a port.

Package graphics-ports

Signature port-height port => result

Arguments port A graphics port.

Values result An integer.

Description The *port-height* function returns the pixel height of *port*.

**port-string-height**

Function

Summary Returns the height of a string drawn to a given port in pixels.

Package graphics-ports

Signature port-string-height port string => height

Arguments port A graphics port.

string A string.

Values height An integer.

Description The *port-string-height* function returns the *height* in pixels of *string* when drawn to *port*. The font used is the one currently in the port’s graphics state.
**port-string-width**

*Function*

**Summary**
Returns the width of a string drawn to a given port in pixels.

**Package**
`graphics-ports`

**Signature**
`port-string-width port string => width`

**Arguments**
- `port` A graphics port.
- `string` A string.

**Values**
- `width` An integer.

**Description**
The `port-string-width` function returns the `width` in pixels of `string` when drawn to `port`. The font used is the one currently in the port’s graphics state.

**Note:** To compute the horizontal extents of each successive character in a string for a given port or font, use `compute-char-extents`.

**See also**
`compute-char-extents`

---

**port-width**

*Function*

**Summary**
Returns the pixel width of a port.

**Package**
`graphics-ports`

**Signature**
`port-width port => width`

**Arguments**
- `port` A graphics port.

**Values**
- `width` An integer.

**Description**
The `port-width` function returns the pixel width of `port`. 
postmultiply-transforms  

**Function**

**Summary**  
Postmultiplies two transforms.

**Package**  
graphics-ports

**Signature**  
postmultiply-transforms transform1 transform2

**Arguments**  
transform1  
A transform.

transform2  
A transform.

**Description**  
The `postmultiply-transforms` function postmultiplies the partial 3 x 3 matrix represented by `transform1` by the partial 3 x 3 matrix represented by `transform2`, storing the result in `transform1`. In the result, the translation, scaling and rotation operations contained in `transform2` are effectively performed after those in `transform1`.

transform1 = transform1 . transform2

premultiply-transforms  

**Function**

**Summary**  
Premultiplies two transforms.

**Package**  
graphics-ports

**Signature**  
premultiply-transforms transform1 transform2

**Arguments**  
transform1  
A transform.

transform2  
A transform.

**Description**  
The `premultiply-transforms` function premultiplies the partial 3 x 3 matrix represented by `transform1` by the partial 3 x 3 matrix represented by `transform2`, storing the result in
In the result, the translation, scaling and rotation operations contained in $\text{transform2}$ are effectively performed before those in $\text{transform1}$.

$$\text{transform1} = \text{transform2} \cdot \text{transform1}$$

### read-and-convert-external-image

**Function**

**Summary**
Returns an image converted from an external image read from a file.

**Package**
graphics-ports

**Signature**
`read-and-convert-external-image gp file &key transparent-color-index => image, external-image`

**Arguments**
- `gp` A CAPI pane.
- `file` A pathname designator.
- `transparent-color-index` An integer or `nil`.

**Values**
- `image` An image.
- `external-image` An external image.

**Description**
Returns an image converted from an external image read from `file`. The external image is returned as a second value.

`transparent-color-index` is interpreted as described for `read-external-image`.

**See also**
- `convert-external-image`
- `external-image`
- `read-external-image`
read-external-image

Function

Summary
Returns an external image read from a file.

Package
graphics-ports

Signature
read-external-image file &key transparent-color-index type

Arguments
file
A pathname designator.

transparent-color-index
An integer or nil.

type
A keyword, or nil.

Values
image
An external image.

Description
The read-external-image function returns an external image read from file.

transparent-color-index specifies the index of the transparent color in the color map. transparent-color-index works only for images with a color map, that is, those with 256 colors or less. The default value is nil, meaning that there is no transparent color.

type tells read-external-image that the image is in a particular graphics format. Currently the only recognised value is :bmp, which means the image is read as a Bitmap. Other values of type cause read-external-image to read the image according to the file type of file. "bmp" or "dib" mean that the image is read as a Bitmap. Other file types are handled in Operating System-specific ways. See the Graphics Ports chapter in the LispWorks CAPI User Guide for details. The default value of type is nil.

Example
To see the effect of transparent-color-index, edit examples/capi/graphics/images.lisp.
Specify a non-white :background for the viewer pane. Use an image editing tool to find the transparent color index (183 in this image) and change the call to read-external-image like this:

\[(\texttt{gp:read-external-image file :transparent-color-index 183})\]

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 graphics-ports

Signature \(\texttt{rectangle-bind \ ((a \ b \ c \ d) \ rectangle) \ &body \ body => result}\)

Arguments
\begin{itemize}
  \item \(a\) A variable.
  \item \(b\) A variable.
  \item \(c\) A variable.
  \item \(d\) A variable.
  \item \(\textit{rectangles}\) A rectangle.
  \item \(\textit{body}\) A body of code.
\end{itemize}

Values \(\textit{result}\) The return value of the last form in \textit{body}.

Description The rectangle-bind macro binds the variables \(a\ b\ c\ d\) to left top right bottom of rectangle for the body of the macro.
**rectangle-bottom**  
*Macro*

**Summary**  
Get and sets the *bottom* element of a rectangle.

**Package**  
`graphics-ports`

**Signature**  
`rectangle-bottom rectangle => bottom`

**Signature**  
`(setf rectangle-bottom) bottom rectangle => bottom`

**Arguments**  
`rectangle`  
A rectangle.

**Values**  
`bottom`  
A real number.

**Description**  
Returns and via `setf` sets the *bottom* element of `rectangle`.

`rectangle` is a list of numbers (*left top right bottom*).

---

**rectangle-height**  
*Macro*

**Summary**  
Returns the *height* element of a rectangle.

**Package**  
`graphics-ports`

**Signature**  
`rectangle-height rectangle => height`

**Arguments**  
`rectangle`  
A rectangle.

**Values**  
`height`  
A real number.

**Description**  
The `rectangle-height` macro returns the difference between the *bottom* and *top* elements of `rectangle`.

`rectangle` is a list of numbers (*left top right bottom*).
**rectangle-left**

*Macro*

**Summary**  Gets and set the *left* element of a rectangle.

**Package**  `graphics-ports`

**Signature**  `rectangle-left rectangle => left`

**Signature**  `(setf rectangle-left) left rectangle => left`

**Arguments**  `rectangle`  A rectangle.

**Values**  `left`  A real number.

**Description**  The `rectangle-left` macro returns and via `setf` sets the *left* element of `rectangle`. `rectangle` is a list of numbers (*left* *top* *right* *bottom*).

---

**rectangle-right**

*Macro*

**Summary**  Gets and sets the *right* element of a rectangle.

**Package**  `graphics-ports`

**Signature**  `rectangle-right rectangle => right`

**Signature**  `(setf rectangle-right) right rectangle => right`

**Arguments**  `rectangle`  A rectangle.

**Values**  `right`  A real number.

**Description**  The `rectangle-right` macro returns and via `setf` sets the *right* element of `rectangle`. `rectangle` is a list of numbers (*left* *top* *right* *bottom*).
rectangle-top  

**Macro**

**Summary**
Gets and sets the top element of a rectangle.

**Package**
graphics-ports

**Signature**
rectangle-top rectangle => top

**Signature**
(setf rectangle-top) top rectangle => top

**Arguments**
rectangle  
A rectangle.

top  
A real number.

**Values**

top

**Description**
The rectangle-top macro returns and via setf sets the top element of rectangle.

rectangle is a list of numbers (left top right bottom).

rectangle-union  

**Function**

**Summary**
Returns the four values representing a union of two rectangles.

**Package**
graphics-ports

**Signature**
rectangle-union left-1 top-1 right-1 bottom-1

left-2 top-2 right-2 bottom-2

=> left, top, right, bottom

**Arguments**
left-1  
A real number.

top-1  
A real number.

right-1  
A real number.

bottom-1  
A real number.

left-2  
A real number.
The `rectangle-union` function returns four values: the left, top, right and bottom 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 ordered-rectangle-union

**rectangle-width**

*Macro*

**Summary**

Returns the difference between the left and right elements of a rectangle.

**Package**

`graphics-ports`

**Signature**

`rectangle-width rectangle => width`

**Arguments**

`rectangle` A rectangle

**Values**

`width` A real number

**Description**

The `rectangle-width` macro returns the difference between right and left elements of rectangle. `rectangle` is a list of numbers (left top right bottom).
rect-bind

Summary
Binds four variables to the elements of a rectangle across a body of code.

Package
graphics-ports

Signature
rect-bind ((x y width height) rectangle) &body body => result

Arguments
x A variable.
y A variable.
width A variable.
height A variable.
rectangle A rectangle.
body A body of Lisp code.

Values
result The return value of the last form in body.

Description
The rect-bind macro binds x y width height to the appropriate values from rectangle and executes the body forms. The rectangle is a list of the form (left top right bottom).

register-image-load-function

Summary
Registers one or more image identifiers with an image loading function.

Package
graphics-ports

Signature
register-image-load-function image-id image-load-function &key image-translation-table

Arguments
image-id An image identifier or a list of image identifiers.
image-load-function

A function.

image-translation-table

An image translation table.

Description

The register-image-load-function function registers one or more image-ids with an image-load-function in the image-translation-table. If image-load-function is nil it causes the default loader to be used in subsequent calls to load-image. The image-id argument can be a list of identifiers or a single identifier. The default value of image-translation-table is *default-image-translation-table*.

See also

*default-image-translation-table*
load-image

register-image-translation

Function

Summary

Registers an image identifier and image loading function with a translation in an image translation table.

Package

graphics-ports

Signature

register-image-translation image-id translation &key
image-translation-table image-load-fn

Arguments

image-id An image identifier.

translation An image translation.

image-translation-table An image translation table.

image-load-fn An image loading function.
Description

The `register-image-translation` function registers `image-id` and `image-load-fn` with the `translation` in the `image-translation-table`. When `load-image` is called with second argument `image-id`, the `image-load-fn` is called with `translation` as its second argument. If `image-load-fn` is `nil`, the image translation table’s default image loader is used; this converts an external image object or file to an image. If `translation` is `nil` the identifier is deregistered. Returns the `image-id` and the `image-load-fn`. The default value of `image-translation-table` is `*default-image-translation-table*`.

See also

* `*default-image-translation-table*`
* `load-image`
* `reset-image-translation-table`

reset-image-translation-table

Function

Summary

Clears the image translation table hash tables.

Package

graphics-ports

Signature

`reset-image-translation-table &key image-translation-table`

Arguments

`image-translation-table`

An image translation table.

Description

The `reset-image-translation-table` function clears the image translation table hash tables and set the default `image-load-fn` to `read-and-convert-external-image`. The default value of `image-translation-table` is `*default-image-translation-table*`.

See also

* `*default-image-translation-table*`
* `read-and-convert-external-image`
* `register-image-translation`
separation

Function

Summary
Returns the distance between two points.

Package
 graphics-ports

Signature
separation x1 y1 x2 y2 => dist

Arguments
x1 An integer.
y1 An integer.
x2 An integer.
y2 An integer.

Values
dist A real number.

Description
The separation function returns the distance between points (x1 y1) and (x2 y2).

set-default-image-load-function

Function

Summary
Sets the default image load function of an image translation table.

Package
 graphics-ports

Signature
set-default-image-load-function image-load-function &key image-translation-table

Arguments
image-load-function
An image load function.

image-translation-table
An image translation function.
Description  The `set-default-image-load-function` function sets the default image load function of `image-translation-table`. The default image load function is `read-and-convert-external-image`. The default value of `image-translation-table` is `*default-image-translation-table*`.

---

**set-graphics-port-coordinates**

**Function**

Summary  Modifies the transform of a port such that the edges of the port correspond to the arguments given.

Package  `graphics-ports`

Signature  

\[\text{set-graphics-port-coordinates} \quad \text{port} \quad \&\text{key} \quad \text{left} \quad \text{top} \quad \text{right} \quad \text{bottom}\]

Arguments  

- `port`  A graphics port.
- `left`  A real number.
- `top`  A real number.
- `right`  A real number.
- `bottom`  A real number.

Description  The `set-graphics-port-coordinates` function modifies the transform of the `port` is permanently such that the edges of the port correspond to the values of the arguments.

Example  The following code

\[
\text{(set-graphics-port-coordinates \text{port} \ :left \ -1.0 \ :top \ 1.0 \ :right \ 1.0 \ :bottom \ -1.0)}
\]

changes the coordinates of the port so that the point (0 0) is in the exact center of the port and the edges are a unit distance away, with a right-handed coordinate system.
By default, left and top are 1.

**set-graphics-state**  
*Function*

**Summary**
Directly alters the graphics state of a graphics port according to the keyword arguments supplied.

**Package**
graphics-ports

**Signature**
```lisp
set-graphics-state port &rest args  
&key transform foreground background  
operation stipple pattern fill-style thickness  
scale-thickness dashed dash line-end-style  
line-join-style mask mask-x mask-y font
```

**Arguments**
- **port** A graphics port.

**Description**
This directly alters the graphics state of `port` according to the values of the keyword arguments `args`. Unspecified keywords leave the associated slots unchanged.

See `make-graphics-state` for valid values for `args`.

**See also**
- `make-graphics-state`
- `with-graphics-state`

**transform**  
*Type*

**Summary**
The transform type, defined for transform objects.

**Package**
graphics-ports

**Description**
The `transform` type is the type defined for transform objects, which are six-element lists of numbers.
transform-area

**Function**

**Summary**
Transforms a set of points and returns the resulting rectangle.

**Package**
graphics-ports

**Signature**
transform-area transform x y width height => rectangle

**Arguments**
- transform: A transform.
- x: A real number.
- y: A real number.
- width: A real number.
- height: A real number.

**Values**
- rectangle: A rectangle.

**Description**
The `transform-area` function transforms the points \((x, y)\) and \((x + \text{width}, y + \text{height})\) and returns the transformed rectangle as \((x, y, \text{width}, \text{height})\) values.

transform-distance

**Function**

**Summary**
Transforms a distance vector by the rotation and scale of a transform.

**Package**
graphics-ports

**Signature**
transform-distance transform dx dy => dx2, dy2

**Arguments**
- transform: A transform.
- dx: A real number.
- dy: A real number.

**Values**
- dx2: A real number.
\( dy2 \)       A real number.

Description  The \texttt{transform-distance} function transforms the distance \((dx \ dy)\) by the rotation and scale in the \texttt{transform}. The translation in the transform is ignored. Transformed \((dx \ dy)\) is returned as two values.

\begin{center}
\textbf{transform-distances}  \hspace{3cm} \textit{Function}
\end{center}

\begin{center}
\textbf{Summary}  Transforms a list of alternating distance vectors by a given transform.
\end{center}

\begin{center}
\textbf{Package}  \texttt{graphics-ports}
\end{center}

\begin{center}
\textbf{Signature}  \texttt{transform-distances transform distances => result}
\end{center}

\begin{center}
\textbf{Arguments}  \ 
\textit{transform}  A transform.
\textit{distances}  A list of pairs of real numbers.
\end{center}

\begin{center}
\textbf{Values}  \ 
\textit{result}  A list of pairs of real numbers.
\end{center}

\begin{center}
\textbf{Description}  The \texttt{transform-distances} function transforms a list of alternating \((dx \ dy)\) pairs in \texttt{distances} by the \texttt{transform}. Transformed values are returned as a new list.
\end{center}

\begin{center}
\textbf{transform-is-rotated}  \hspace{3cm} \textit{Function}
\end{center}

\begin{center}
\textbf{Summary}  Returns \texttt{t} if a given transform contains a rotation.
\end{center}

\begin{center}
\textbf{Package}  \texttt{graphics-ports}
\end{center}

\begin{center}
\textbf{Signature}  \texttt{transform-is-rotated transform => bool}
\end{center}

\begin{center}
\textbf{Arguments}  \ 
\textit{transform}  A transform.
\end{center}
Values | bool | A boolean.

Description | The `transform-is-rotated` function returns `t` if `transform` contains any rotation.

---

**transform-point**

*Function*

**Summary**
Transforms a point by multiplying it by a transform.

**Package**
gegraphics-ports

**Signature**
`transform-point` `transform` `x` `y` `=>` `xnew` `ynew`

**Arguments**
- `transform` A transform.
- `x` A real number.
- `y` A real number.

**Values**
- `xnew` A real number.
- `ynew` A real number.

**Description**
The `transform-point` function transforms the point `(x y)` by multiplying it by `transform`. The transformed `(x y)` is returned as two values.

---

**transform-points**

*Function*

**Summary**
Transforms a list of points by a transform.

**Package**
gegraphics-ports

**Signature**
`transform-points` `transform` `points` &optional `into` `=>` `result`

**Arguments**
- `transform` A transform.
The `transform-points` function transforms a list of alternating \((x\ y)\) pairs in \(points\) by the \(transform\). If \(into\) is supplied it is modified to contain the result and must be a list the same length as \(points\). If \(into\) is not supplied, a new list is returned.

### transform-rect

**Summary**

Returns the transform of two points representing the top-left and bottom-right of a rectangle.

**Package**

`graphics-ports`

**Signature**

\[
\text{transform-rect } \text{transform } \text{left } \text{top } \text{right } \text{bottom} \Rightarrow \text{left2 } \text{top2 } \text{right2 } \text{bottom2}
\]

**Arguments**

- \(transform\) A transform.
- \(left\) A real number.
- \(top\) A real number.
- \(right\) A real number.
- \(bottom\) A real number.

**Values**

- \(left2\) A real number.
- \(top2\) A real number.
- \(right2\) A real number.
- \(bottom2\) A real number.
2  GP Reference Entries

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*

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

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.
unlless-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  graphics-ports

Signature
unlless-empty-rect-bind ((x y width height) rectangle)  
&body body => result

Arguments
x  A variable.
y  A variable.
width  A variable.
height  A variable.
rectangle  A rectangle.
body  A body of Lisp code.

Values
result  The return value of the last form executed in body.

Description
The unless-empty-rect-bind macro binds x, y, width, and height to the appropriate values from rectangle and if the width and height are both positive, executes the body forms.

untransform-distance  

Function

Summary
Transforms a distance by the rotation and scale of the inverse of a given transform.

Package  graphics-ports

Signature
untransform-distance transform dx dy => x, y

Arguments
transform  A transform.
The `untransform-distance` function transforms the distance \((dx \ dy)\) by the rotation and scale of the effective inverse of \(transform\). The translation in the inverse transform is ignored. The transformed distance \((dx \ dy)\) is returned as two values.

**untransform-distances**  
*Function*

**Summary**  
Transforms a list of integer pairs representing distances by the inverse of a transform.

**Package**  
`graphics-ports`

**Signature**  
`untransform-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 `untransform-distances` function transforms a list of alternating \((dx \ dy)\) pairs in `distances` by the effective inverse of `transform`. 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.
untransform-point

Function

 Summary  Transforms a list of points by the inverse of a given transform.

 Package  graphics-ports

 Signature  untransform-points transform points &optional into => result

 Arguments  transform  A transform.
  points  A list of pairs of real numbers.
  into  A list.

 Values  result  A list of pairs of real numbers.

 Description  The untransform-points function transforms a list of alternating (x y) pairs in points by the effective inverse of transform. If into is supplied it must be a list the same length as points. If into 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**
graphics-ports

**Signature**
validate-rectangle object &optional x y width height => result

**Arguments**

- **object**
  An instance of a subclass of graphics-ports-mixin or a subclass of pinboard-object.

- **x**
  A real number.

- **y**
  A real number.

- **width**
  A real number.

- **height**
  A real number.

**Values**

- **result**
  A boolean.

**Description**
The given area of object 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 &optional arguments.

The result is non-nil if the function succeeds and nil if it fails (doing nothing).

**Note:** this function is not fully implemented on all platforms.

On Windows, it succeeds for all valid values of x, y, width and height.

On Cocoa, it fails if x, y, width and height are passed.

On Motif, it fails in all cases.

**See also**
invalidate-rectangle
with-dither

**Macro**

**Summary**
Specifies a dither for use within a specified body of code.

**Package**
graphics-ports

**Signature**

```
with-dither (dither-or-size) &body body => result
```

**Arguments**
- `dither-or-size` See Description.
- `body` A body of Lisp code.

**Values**
- `result` The return value of the last form executed in `body`.

**Description**
The `with-dither` function specifies a dither for use within `body`. The `dither-or-size` argument can be a dither mask object from `make-dither` or a size, in which case a dither of that size is created.

**Note:** Dithers do not affect drawing or the antialiasing that occurs when drawing in Cocoa.

**See also**
dither-color-spec
make-dither
initialize-dithers

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**
graphics-ports

**Signature**

```
with-graphics-mask (port mask mask-x mask-y &key) &body body => result
```

**Arguments**
- `port` A graphics port.
mask A list of the form \((x \ y \ \text{width} \ \text{height})\) or \text{nil}.

mask-x An integer.

mask-y An integer.

body A body of Lisp code.

Values result The return value of the last form executed in body.

Description The \texttt{with-graphics-mask} macro binds the mask slot of port’s graphic state to a rectangular area across the execution of body.

By default, mask-x and mask-y are both 0. These values are used only on Motif.

Example For a mask value of \((x \ y \ \text{width} \ \text{height})\) drawing is limited to the rectangular region whose X coordinate is in the range

\[
\text{mask-x} + x \ \text{to} \ \text{mask-x} + x + \text{width}
\]

and whose Y coordinate is in the range

\[
\text{mask-y} + y \ \text{to} \ \text{mask-y} + y + \text{height}
\]

\textit{with-graphics-rotation} \quad \text{Macro}

Summary Performs a call to \texttt{apply-rotation} with a given angle for the duration of the macro’s body.

Package \texttt{graphics-ports}

Signature \texttt{with-graphics-rotation (port angle) \&body body => result}

Arguments port A graphics port.

angle A real.

body A body of Lisp code.
Values

\textit{result} The return value of the last form executed in \textit{body}.

Description

The \texttt{with-graphics-rotation} macro performs a call to
\begin{verbatim}
(apply-rotation transform angle)
\end{verbatim}
on the port’s transform for the duration of the body of the macro.

See also \texttt{apply-rotation}

\textbf{with-graphics-scale} \textit{Macro}

Summary

Performs a call to \texttt{apply-scale} with a given scale for the duration of the macro’s body.

Package \texttt{graphics-ports}

Signature

\texttt{with-graphics-scale (port sx sy) \&body body => result}

Arguments

\texttt{port} A graphics port.

\texttt{sx} A real number.

\texttt{sy} A real number.

\texttt{body} A body of Lisp code.

Values

\textit{result} The return value of the last form executed in \textit{body}.

Description

The \texttt{with-graphics-scale} macro performs a call to
\begin{verbatim}
(apply-scale transform sx sy)
\end{verbatim}
on the port’s transform for the duration of the body of the macro.

See also \texttt{apply-scale}
**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**
graphics-ports

**Signature**

\[
\text{with-graphics-state} (\text{\textit{port} &rest \textit{args} \&key transform foreground background operation thickness scale-thickness dashed dash line-end-style line-joint-style mask font state fill-style stipple pattern mask-x mask-y) body => result}
\]

**Arguments**

- **port**
A graphics port.

- **body**
A body of Lisp code.

**Values**

- **result**
The return value of the last form executed in \textit{body}.

**Description**

The **\texttt{with-graphics-state}** macro binds the graphics state values for the specified port to the values specified in the \textit{args} list. The keyword arguments \textit{args} correspond to the slots in the graphics state, as described in **\texttt{set-graphics-state}**. See **\texttt{make-graphics-state}** for valid values for \textit{args}.

For example:

\[
\text{(with-graphics-state (port :thickness 12 :foreground fore-color) ...)}
\]

Arguments that are not supplied default to the current state of that slot in the graphics state. The arguments \texttt{fill-style}, \texttt{stipple}, \texttt{pattern}, \texttt{mask-x} and \texttt{mask-y} are used only on Unix.

An extra keyword argument :\texttt{state} can be used. The value must be a graphics state object created by a call to **\texttt{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
port A graphics port.
transform A transform.
body A body of Lisp code.

Values
result The return value of the last form executed in
  body.

Description
The with-graphics-transform macro combines the trans-
form associated with the graphics port port with transform
during the body of the macro. The port is given a new trans-
form obtained by pre-multiplying its current transform with
transform. This has the effect of preceding any translation, scal-
ing 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**  
graphics-ports

**Signature**  
with-graphics-translation (port dx dy) &body body => result

**Arguments**  
- **port**: A graphics port.
- **dx**: A real number.
- **dy**: A real number.
- **body**: A body of Lisp code.

**Values**  
- **result**: The return value of the last form executed in body.

**Description**  
The `with-graphics-translation` macro performs a call to (apply-translation transform dx dy) on the port’s transform for the duration of body 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**  
graphics-ports

**Signature**  
with-inverse-graphics (port) &body body => result

**Arguments**  
- **port**: A graphic port.
- **body**: A body of Lisp code.
The return value of the last form executed in body.

The with-inverse-graphics macro ensures that all drawing function calls to port within the body of the macro are executed with the foreground and background slots of the graphics state of the port swapped around.

**without-relative-drawing**  
*Macro*

Summary  Evaluates a body of Lisp code with the relative and collect internal variables of the port set to nil.

Package  graphics-ports

Signature  without-relative-drawing (port) &body body => result

Arguments  
- **port**  A graphic port.
- **body**  A body of Lisp code.

Values  
- **result**  The return value of the last form executed in body.

Description  The with-relative-drawing macro evaluates the code in body with the relative and collect internal variables of the pixmap graphics port port set to nil to turn off the port’s collecting of drawing bounds and automatic shifting of its origins. Use this macro only within a with-pixmap-graphics-port 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.
with-pixmap-graphics-port (port pane width height &key
  background collect relative clear)
&body body) => result

Arguments

port       A graphic port.
pane       An output pane.
width      An integer.
height     An integer.
background A color keyword.
collect    A boolean.
relative   A boolean.
clear      A list or t.
body       A body of Lisp code.

Values

result     The return value of the last form executed in body.

Description

The with-pixmap-graphics-port macro binds port to a new pixmap graphics-port. pane and the other arguments are passed to create-pixmap-port. The body is then evaluated. The port is destroyed when body 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 graphics-ports
with-transformed-area

**Signature**

with-transformed-area (points port left top right bottom)
&body body

**Arguments**

points
A variable.

port
A graphics port.

left
A real number.

top
A real number.

right
A real number.

bottom
A real number.

body
A body of Lisp code.

**Values**

result
The return value of the last form executed in body.

**Description**

The with-transformed-area macro transforms a rectangle, binding the resulting four corner points to points for the duration of body. The left top right bottom values represent a rectangular area bounded by four points. The four points are transformed by the port’s transform and the list of eight values (alternating x and y values for four points) bound to the points variable for the duration of the macro body.

**with-transformed-point**

**Macro**

**Summary**

Binds a point transformed by a given port’s transform to two variables across the body of the macro.

**Package**

graphics-ports

**Signature**

with-transformed-point (new-x new-y port x y) &body body
=> result

**Arguments**

new-x
A variable.

new-y
A variable.
The `with-transformed-points` macro binds `points` to a new list of `x` and `y` values obtained by post-multiplying them by the current transform of `port`, and then evaluates `body`. The `points` symbol must be bound to a list of alternating `x` and `y` values representing coordinate points in the `port`. 

**Synopsis**

```
with-transformed-points (points port) &body body => result
```

---

**Argument**

- `points`: A list of real numbers.
- `port`: A graphics port.

**Values**

- `result`: The return value of the last form executed in `body`. 

---

**Description**

The `with-transformed-points` macro transforms the point given by `(x y)` using the `port`'s transform and the resulting values are bound to the `new-x` and `new-y` variables. The `body` of the macro is then evaluated with this binding.
**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**
graphics-ports

**Signature**
```lisp
with-transformed-rect (nx1 ny1 nx2 ny2 port x1 y1 x2 y2)  
&body body => result
```

**Arguments**
- `nx1`: A variable.
- `ny1`: A variable.
- `nx2`: A variable.
- `ny2`: A variable.
- `port`: A graphics port.
- `x1`: A real number.
- `y1`: A real number.
- `x2`: A real number.
- `y2`: A real number.
- `body`: A body of Lisp code.

**Values**
- `result`: The return value of the last form executed in `body`.

**Description**
During the evaluation of the `with-transformed-rect` macro `body`, the two points `(x1, y1)` and `(x2, y2)` are transformed by the port’s current transform and the resulting values bound to the variables named by the `nx1 ny1 nx2 ny2` args.

**write-external-image**  
*Function*

**Summary**
Writes external image data to a file.
Package  graphics-ports
Signature  write-external-image external-image file &key if-exists
Arguments  external-image  An external-image.
           file  A file.
           if-exists  A keyword.
Description  The write-external-image function writes an external image to a file file. It writes the image data byte-for-byte without attempting any conversion of the image format.

if-exists is passed to open when opening file. The default value of if-exists is :error.
See also  externalize-image
2  GP 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**

color

**Signature**

`apropos-color-alias-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 aliases in the color-database defining color aliases. By convention these are in the keyword package.
Example

In this example, a color alias is defined for the color `indianred1`. `apropos-color-alias-names` only returns this alias, rather than both the alias and the original color, despite the similarity in the names.

```
COLOR 8 > (define-color-alias :myindianred1
    :indianred1)
(#S(COLOR-ALIAS COLOR :INDIANRED1))
COLOR 9 > (apropos-color-names "INDIANRED1")
(,:INDIANRED1 :MYINDIANRED1)
COLOR 10 > (apropos-color-alias-names "INDIANRED1")
(,:MYINDIANRED1)
COLOR 11 >
```

See also

`apropos-color-names`
`apropos-color-spec-names`
`get-all-color-names`

---

**apropos-color-names**

*Function*

**Summary**

Returns colors and color aliases containing a given string.

**Package**

`color`

**Signature**

`apropos-color-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 present in the color-database defining color aliases. By convention these are in the keyword package.
Example>

COLOR-4> (color:apropos-color-names "RED")

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 > (apropos-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.
This should contain definitions for all the colors used in the environment when you start it. Those colors are determinable from the file `config/colors.db`.

To replace the current color database with a new one, do the following:

```lisp
(setf color:*color-database* (color:make-color-db))
```

See also
- delete-color-translation
- read-color-db
- load-color-database

**color-level**

*Function*

Returns the gray level of a color specification.

**Package**
color

**Signature**
color-level color-spec => gray-level

**Arguments**
color-spec A color specification.

**Values**
gray-level Color component from the :gray model.

**Description**
Return the gray level of color-spec. If color-spec is not from the :GRAY model, the component is calculated.

```
COLOR 2 > (color:make-gray 0.66667s0)
#(:GRAY 0.66667S0)

COLOR 3 > (color-level *)
0.66667S0

COLOR 4 >
```
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 0.6666780)

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
**COLOR Reference Entries**

**Signature**

```lisp
(colors= color1 color2 &optional tolerance => bool)
```

**Arguments**
- `color1`: A color specification.
- `color2`: A color specification.
- `tolerance`: A tolerance level within which `color1` and `color2` may vary. The default value is `0.001s0`.

**Values**
- `bool`: `t` if the two colors are equal within the given tolerance, `nil` otherwise.

**Description**
Return `t` if the two colors are equal to the given tolerance.

**See also**
- `ensure-<command>`
- `convert-color`

---

**convert-color**

**Summary**
Return the representation of a color specification on a given graphics port.

**Package**
- `color`

**Signature**

```lisp
(convert-color port color &key errorp => color-rep)
```

**Arguments**
- `port`: A graphics port.
- `color`: A color specification.
- `errorp`: If `t`, check for errors. By default, this is `t`.

**Values**
- `color-rep`: Representation of `color` on `port`.

**Description**
Return the representation of `color` on the given graphics port `port`. 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 \texttt{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
\begin{itemize}
\item \texttt{colors=}
\item \texttt{ensure-<command>}
\item \texttt{unconvert-color}
\end{itemize}

\textbf{define-color-alias} \hspace{1cm} \textit{Function}

\textbf{Summary} \hspace{1cm} Lets you define an alias for a color specification or alias.

\textbf{Package} \hspace{1cm} \texttt{color}

\textbf{Signature} \hspace{1cm} \texttt{define-color-alias name color &optional if-exists => alias}

\textbf{Arguments} \hspace{1cm}
\begin{itemize}
\item \textit{name} \hspace{1cm} The name of the new alias.
\item \textit{color} \hspace{1cm} A color specification for the new alias.
\item \textit{if-exists} \hspace{1cm} This can be one of the following:
  \begin{itemize}
  \item \texttt{:replace} — Replace any existing alias.
  \item \texttt{:error} — Raise an error if alias is already defined.
  \item \texttt{:ignore} — Ignore redefinition of an alias.
  \end{itemize}
\end{itemize}

By default, it is \texttt{:replace}.

\textbf{Values} \hspace{1cm} \textit{alias} \hspace{1cm} The color alias.

\textbf{Description} \hspace{1cm} Define \textit{name} to be a color alias for \textit{color}, which may be another color alias or a color-spec.

\textbf{Example 1} \hspace{1cm} \begin{verbatim}
COLOR 16 > (define-color-alias :mygray :darkslategray)
(#S(COLOR-ALIAS COLOR :DARKSLATEGRAY))
\end{verbatim}
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 :DARKSLATEGRA
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
getc-color-spec

define-color-models

Macro

Summary
Defines all the color models.

Package
color

Signature
define-color-models model-descriptors=> color-models

Arguments
model-descriptors A list, each element being a model-descriptor.

Values
color-models The color models defined.

Description A model descriptor has the syntax:
A component-descriptor 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))
                       (:cmyk (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**
color

**Signature**
`delete-color-translation color-name => <no values>`

**Arguments**
`color-name` A defined color spec or alias.
3 COLOR Reference Entries

Values
None.

Description
Both original entries and aliases can be removed.

See also
load-color-database
*color-database*
read-color-db

ensure-<command>

Function

Summary
Return a color specification for a given model. The model depends on the particular function called

Package
color

Signature
ensure-rgb color-spec => result
ensure-hsv color-spec => result
ensure-gray color-spec => result
ensure-model-color color-spec model => result
ensure-color color-spec match-color-spec => result

Arguments
For all functions:

color-spec A color specification.
match-color-spec A color specification.
model A color-model (:rgb, :hsv or :gray).

Values
result A color specification.

Description
These functions all return a color specification, given (at least) a color specification as argument.

ensure-rgb, ensure-hsv and ensure-gray all return a color specification in the appropriate model. If color-spec is in the same model, it is just returned. Otherwise a new color specifi-
cation for that model is calculated. Thus, `ensure-rbg` 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
get-all-color-names &optional sort => color-names

Arguments:
- **sort**: If `t`, sort list of color names alphabetically. By default, this is `nil`.

Values:
- **color-names**: 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 alphabetically sorted on the symbol names if the optional argument is non-nil.

See also:
- `apropos-color-names`
- `apropos-color-spec-names`
- `apropos-color-alias-names`

get-color-alias-translation

**Function**

Summary:
Return the ultimate color name associated with `color-alias`.

Package:
color

Signature:
get-color-alias-translation color-alias => color-name

Arguments:
- **color-alias**: A defined color alias.

Values:
- **color-name**: The color name associated with `color-alias`.

Example:
COLOR 23 > (color:define-color-alias :lispworks-blue (color:make-rgb 0.70s0 0.90s0 0.99s0))
(#S(COLOR-ALIAS COLOR #(:RGB 0.699999S0 0.9S0 0.99S0)))
COLOR 24 > (color:define-color-alias :color-background :lispworks-blue)
(#S(COLOR-ALIAS COLOR :LISPWORKS-BLUE))
get-color-spec

Summary
Returns the color-spec for a color.

Package
color

Signature
get-color-spec color => color-spec

Arguments
color
A defined color specification, color alias, or an original color name.

Values
color-spec
A color specification.

Description
Returns the color-spec for color, which can be a color-spec, a color-alias, or an original color name.

Example
COLOR 28 > (color:define-color-alias :lispworks-blue
          (color:make-rbg 0.70s0 0.90s0 0.99s0))
          (#S(COLOR-ALIAS COLOR #(:RGB 0.699999S0 0.9S0 0.99S0)))
COLOR 29 > (color:define-color-alias
          :color-background :lispworks-blue)
          (#S(COLOR-ALIAS COLOR :LISPWORKS-BLUE))
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  color

Signature  load-color-database data => <no values>

Arguments  data A description of a color database.

Values  None.

Description  This loads the color database with color definitions contained
   in data, which should have been obtained via the functions
   color:read-color-db. 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
Function

make-hsv

Summary
Returns a color specification in the hue-saturation-value model.

Package
color

Signature
make-hsv hue saturation value &optional alpha => color-spec

Arguments
hue A hue component.
saturation A saturation component.
value A value component.
alpha A number between 0 and 1, or nil.

Values
color-spec A color specification.

Description
Return a color-spec in the :HSV model with components hue, saturation and value.

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
COLOR 27 > (color:make-hsv 1.2s0 0.5s0 0.9s0)
#(:HSV 1.2S0 0.5S0 0.9S0)
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
red A red component.
green A green component.
blue A blue component.
alpha A number between 0 and 1, or nil.

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
3 COLOR Reference Entries
Index

A
abort-callback function 1
abort-dialog function 2
abort-exit-chooser 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
director-pane-buffer 141
director-pane-change-callback 132
director-pane-enabled 132
director-pane-fixed-fill 132
director-pane-line-wrap-face 132
director-pane-line-wrap-marker 132
director-pane-text 132
director-pane-wrap-style 132
element-interface 146
element-parent 146
element-widget-name 146
filled 152, 435
filtering-layout-matches-text 159
callbacks 27
for button panels 20
for buttons 17
passing different variables 12
callbacks 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
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
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
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 accessor 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
dock-pane 129
dock-pane 131
element 144
eclipse 152
expanded-item-pinboard-object 157
extended-selection-tree-view 158
external-image 632
filtering-layout 158
foreign-owned-interface 167
form-layout 168
docking-pane 63
graph-edge 176
drawing-pane 136
view-pane 48
instance 36
interface 88
layout 94
interface 108
menu-bar 45
menus 80
options 88
source-interfaces 88
class options
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
dock-pane 129
docking-pane 131
element 144
eclipse 152
expanded-item-pinboard-object 157
extended-selection-tree-view 158
external-image 632
filtering-layout 158
foreign-owned-interface 167
form-layout 168
docking-pane 63
graph-edge 176
drawing-pane 136
view-pane 48
instance 36
interface 88
layout 94
interface 108
menu-bar 45
menus 80
options 88
source-interfaces 88
class options
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
dock-pane 129
docking-pane 131
element 144
eclipse 152
expanded-item-pinboard-object 157
extended-selection-tree-view 158
external-image 632
filtering-layout 158
foreign-owned-interface 167
form-layout 168
docking-pane 63
graph-edge 176
graph-node 176
collection-items accessor function 53
collection-items-count-function accessor function 53
collection-items-get-function accessor function 53
collection-items-map-function accessor function 53
collection-last-search generic function 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 initarg 246
color-level function 720, 722
color-model function 723
colors= function 723
color-screen class 59
:column initarg 188
:column-function initarg 310
column-layout class 60
column-layout-divider 61
:columns initarg 310
:combine-child-constraints initarg 516, 519
:command initarg 493
command table 341
complete-button image identifier 534
:complete-do-action initarg 525
:completion item in :buttons initarg 531
:completion-function initarg 524
component-name function 63
:component-name initarg 332, 335
compress-external-image function 610
compute-char-extends function 611
comtab 341
Confirm Before Exiting 64, 470
:confirm-change-function initarg 525
:confirm-destroy-function initarg 204
confirm-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 initarg 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 function 72
:create-callback initarg 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 initarg 498

D

dash graphics state parameter 668
dashed graphics state parameter 668
:data initarg 238
:default initarg 242
:default-button initarg 21
*default-editor-pane-line-wrap-marker* variable 76
:default-image-set initarg 552, 560
*default-image-translation-table* variable 616, 656
default-library function 77
:default-p initarg 15
:default-toolbar-states initarg 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 initarg 456
:description initarg 242, 519
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

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-tooltip 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-confirm 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-confirm 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
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-confirm 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-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
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
unhighlight-pinboard-object 582
update-interface-title 584

genericslot
%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-encoding-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-dotted function 669
graphics-state-fill-style function 669
graphics-state-line-end-style function 669
graphics-state-line-joint-style function 669
graphics-state-line-join-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 function 669

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-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
temp generic function 239
item-pinboard-object class 239
item-print-function accessor function 238
:items-accessor function initarg 52, 103, 158, 290, 295, 519, 577
:items-count-function initarg 52, 158, 577
:items-get-function initarg 52, 158, 577
:items-map-function initarg 53, 158, 577
item-selected accessor function 238
:items-function initarg 290, 295
:items-accessor function initarg 290, 295, 519, 577
:items-get-function initarg 290, 295, 519, 577
item-text accessor function 238
K
:keep-selection-p initarg 35
key press event handler 342
:keep-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 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 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
undefined-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
with-transformed-rectangle 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
:mmemonic initarg 16, 21, 290, 297
:mmemonic-escape initarg 16, 21, 290, 297
:mmemonic-text initarg 16, 21
:mmemonic-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 function 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-item-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-get-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

pixblt function 676
pixmap-port class 677
play-sound function 365
:plist initarg 31
:popup-callback initarg 303, 337, 523
popup-confirm function 377
:popup-interface initarg 555
popup-menu-button class 388
popup-menu-button-menu 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 initarg 533
postmultiply-transforms function 680
*ppd-directory* variable 397
premultiply-transforms function 680
:press-callback initarg 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
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
:selected-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
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 492
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 483
set-scroll-position generic function 492
set-scroll-range generic function 493
set-selection function 487
set-text-input-pane-selection function 490
set-top-level-interface-geometry generic function 491
:setup-callback-argument initarg 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 initarg 510, 512
simple-layout class 496
simple-pane class 497
simple-pane-background accessor function 499
simple-pane-cursor 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-horiz-icall function 499
simple-pane-scroll-callback accessor function 499
simple-pane-verti-ical-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 initarg 332
slider class 510
slider-show-value-p accessor function 510
slider-start-point accessor function 510
:slug-end initarg 432
:slug-start initarg 432
:small-image-height initarg 257
:small-image-width initarg 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-initarg 432, 542
  :start-gc-monitor function 513
  :start-point initarg 510
  :start-x initarg 244
  :start-y initarg 244
  :state-image-function initarg 256, 569
  :state-image-height initarg 257, 570
  :state-image-width initarg 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 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-change-callback accessor function 525
text-input-pane-caret-position accessor function 525
text-input-pane-change-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-inplace-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-function initarg 546
:title-gap initarg 168, 546
:title-pane class 543
title-pane-text accessor function 544
:title-position initarg 546
:title-pane class 542
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
Unconvert-color function 737
Undefine-font-alias function 698
Unhighlight-pinboard-object generic function 582
:uniform-size-p initarg 60, 454
Unmap-typeout function 583
Untransform-distance function 700
Untransform-dimensions function 701
Untransform-point function 701
Update-all-interface-titles function 584
Update-interface-title generic function 584
Update-pinboard-object function 585
*update-screen-interfaces-hooks* variable 586
Update-screen-interface-titles function 585
Update-toolbar function 586
:use-images initarg 570
:use-large-images initarg 257
:use-component initarg 332
:use-small-images initarg 257
:use-state-images initarg 257, 570

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

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