Contents

Preface   xxiii

1 CAPI Reference Entries   1
 abort-callback 1
 abort-dialog 2
 abort-exit-confirmers 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
apply-in-pane-process-if-alive 9
arrow-pinboard-object 10
attach-interface-for-callback 11
attach-simple-sink 12
attach-sink 13
beep-pane 14
browser-pane 15
browser-pane-navigate 23
browser-pane-busy 23
browser-pane-go-forward 23
browser-pane-go-back 23
browser-pane-stop 23
browser-pane-refresh 23
browser-pane-property-get 25
browser-pane-property-put 25
button 25
button-panel 31
calculate-constraints 36
calculate-layout 37
callbacks 38
call-editor 41
can-use-metafile-p 42
capi-object 42
capi-object-property 43
check-button 44
check-button-panel 45
choice 46
choice-selected-item 50
choice-selected-item-p 52
choice-selected-items 52
choice-update-item 54
clipboard 55
clipboard-empty 57
clone 57
cocoa-default-application-interface 58
cocoa-view-pane 62
cocoa-view-pane-view 63
collect-interfaces 64
display-popup-menu 115
display-replacable-dialog 116
display-tooltip 117
docking-layout 118
docking-layout-pane-docked-p 122
docking-layout-pane-visible-p 122
document-container 123
document-frame 124
double-headed-arrow-pinboard-object 128
double-list-panel 129
drag-pane-object 131
draw-metafile 133
draw-metafile-to-image 134
drawn-pinboard-object 136
draw-pinboard-object 137
draw-pinboard-object-highlighted 138
draw-pinboard-object-unhighlighted 138
drop-object-allows-drop-effect-p 139
drop-object-collection-index 139
drop-object-collection-item 141
drop-object-drop-effect 142
drop-object-get-object 143
drop-object-pane-x 144
drop-object-pane-y 144
drop-object-provides-format 145
*echo-area-cursor-inactive-style* 145
echo-area-pane 146
*editor-cursor-color* 146
*editor-cursor-active-style* 147
*editor-cursor-drag-style* 147
*editor-cursor-inactive-style* 148
editor-pane 148
editor-pane-blink-rate 158
editor-pane-buffer 159
*editor-pane-composition-selected-range-face-plist* 160
editor-pane-default-composition-callback 160
*editor-pane-default-composition-face* 162
editor-pane-native-blink-rate 163
editor-pane-selected-text 163
editor-pane-selected-text-p 164
editor-pane-stream 164
editor-window 165
element 165
element-container 172
element-interface-for-callback 173
element-screen 174
eclipse 174
ensure-area-visible 174
ensure-interface-screen 175
execute-with-interface 176
execute-with-interface-if-alive 177
exit-confirm 178
exit-dialog 179
expandable-item-pinboard-object 180
extended-selection-tree-view 180
filtering-layout 181
filtering-layout-match-object-and-exclude-p 185
find-graph-edge 185
find-graph-node 186
find-interface 187
find-string-in-collection 188
force-screen-update 188
force-update-all-screens 189
foreign-owned-interface 189
form-layout 190
free-metafile 191
free-sound 192
get-collection-item 192
get-constraints 193
get-horizontal-scroll-parameters 193
get-page-area 194
get-printer-metrics 195
get-scroll-position 196
get-vertical-scroll-parameters 197
graph-edge 198
graph-node 199
graph-node-children 199
graph-object 200
graph-pane 200
graph-pane-add-graph-node 204
graph-pane-delete-object 205
graph-pane-delete-objects 205
graph-pane-delete-selected-objects 206
graph-pane-direction 207
graph-pane-edges 208
graph-pane-nodes 208
graph-pane-object-at-position 209
graph-pane-select-graph-nodes 209
graph-pane-update-moved-objects 210
grid-layout 211
hide-interface 216
hide-pane 216
highlight-pinboard-object 217
image-list 217
image-pinboard-object 219
image-set 220
install-postscript-printer 221
installed-libraries 223
interactive-pane 224
interactive-pane-execute-command 226
interface 227
interface-customize-toolbar 244
interface-display 245
interface-display-title 246
interface-document-modified-p 247
interface-editor-pane 248
interface-extend-title 248
interface-geometry 249
interface-iconified-p 250
interface-keys-style 250
interface-match-p 253
interface-menu-groups 254
interface-preserve-state 255
interface-preserving-state-p 255
interface-reuse-p 256
interface-toolbar-state 257
interface-visible-p 259
interpret-description 260
invalidate-pane-constraints 261
invoke-pane-constraints 261
invoke-untranslated-command 262
item 262
itemp 264
item-pane-interface-copy-object 264
item-pinboard-object 266
labelled-arrow-pinboard-object 267
labelled-line-pinboard-object 267
layout 268
line-pinboard-object 270
line-pinboard-object-coordinates 271
list-panel 272
list-panel-enabled 283
list-panel-filter-state 284
list-panel-items-and-filter 285
list-panel-search-with-function 286
list-panel-unfiltered-items 288
list-view 288
listener-pane 294
listener-pane-insert-value 295
load-cursor 295
load-sound 299
locate-interface 300
lower-interface 301
make-container 302
make-docking-layout-controller 303
make-foreign-owned-interface 303
make-general-image-set 305
make-icon-resource-image-set 306
make-image-locator 307
make-menu-for-pane 307
make-pane-popup-menu 309
make-resource-image-set 311
make-scaled-general-image-set 312
make-scaled-image-set 313
make-sorting-description 314
manipulate-pinboard 315
map-collection-items 318
map-pane-children 319
map-pane-descendant-children 322
map-typeout 323
*maximum-moving-objects-to-track-edges* 323
menu 324
menu-component 328
menu-item 331
menu-object 337
merge-menu-bars 340
message-pane 342
modify-editor-pane-buffer 342
mono-screen 343
move-line 343
multi-column-list-panel 344
multi-line-text-input-pane 349
non-focus-list-interface 350
non-focus-list-toggle-enable-filter 350
non-focus-list-toggle-filter 351
non-focus-list-add-filter 351
non-focus-list-remove-filter 351
non-focus-maybe-capture-gesture 351
non-focus-terminate 353
non-focus-update 354
ole-control-add-verbs 354
ole-control-close-object 355
ole-control-component 356
ole-control-doc 358
ole-control-frame 359
ole-control-i-dispatch 360
ole-control-insert-object 361
ole-control-ole-object 361
ole-control-pane 362
ole-control-pane-frame 365
ole-control-pane-simple-sink 365
ole-control-user-component 366
option-pane 367
output-pane 371
over-pinboard-object-p 384
page-setup-dialog 385
pane-adjusted-offset 386
pane-adjusted-position 387
pane-close-display 388
pane-descendant-child-with-focus 389
pane-got-focus 390
pane-has-focus-p 390
pane-initial-focus 391
pane-interface-copy-object 392
pane-interface-copy-p 392
pane-interface-cut-object 392
pane-interface-cut-p 392
pane-interface-deselect-all 392
pane-interface-deselect-all-p 392
pane-interface-paste-object 392
pane-interface-paste-p 392
pane-interface-select-all 392
pane-interface-select-all-p 392
pane-interface-undo 392
pane-interface-undo-p 392
pane-popup-menu-items 393
pane-screen-internal-geometry 396
pane-string 397
pane-supports-menus-with-images 398
parse-layout-descriptor 398
password-pane 399
play-sound 400
pinboard-layout 401
pinboard-object 404
pinboard-object-at-position 408
pinboard-object-graphics-arg 409
pinboard-object-overlap-p 410
pinboard-pane-position 411
pinboard-pane-size 412
popup-confirmer 413
popup-menu-button 424
*ppd-directory* 424
print-capi-button 425
print-collection-item 425
print-dialog 427
print-editor-buffer 428
print-file 429
print-rich-text-pane 430
print-text 431
printer-configuration-dialog 431
printer-metrics 432
printer-port-handle 434
printer-port-supports-p 434
*printer-search-path* 435
process-pending-messages 436
progress-bar 437
prompt-for-color 437
prompt-for-confirmation 438
prompt-for-directory 440
prompt-for-file 442
prompt-for-files 445
prompt-for-font 446
prompt-for-form 447
prompt-for-forms 449
prompt-for-integer 450
prompt-for-items-from-list 451
prompt-for-number 452
prompt-for-string 453
prompt-for-symbol 455
prompt-for-value 457
prompt-with-list 458
prompt-with-list-non-focus 461
prompt-with-message 466
push-button 467
push-button-panel 469
quit-interface 470
radio-button 472
radio-button-panel 473
raise-interface 474
range-pane 475
range-set-sizes 476
read-sound-sizes 477
rectangle 478
redisplay-collection-item 478
redisplay-interface 479
redisplay-menu-bar 479
redraw-pinboard-layout 480
redraw-pinboard-object 481
reinitialize-interface 481
remove-capi-object-property 482
remove-items 483
replace-dialog 484
replace-items 484
report-active-component-failure 485
reuse-interfaces-p 487
rich-text-pane 487
rich-text-pane-character-format 490
rich-text-pane-operation 491
rich-text-pane-paragraph-format 495
rich-text-version 496
right-angle-line-pinboard-object 496
row-layout 497
screen 499
screen-active-interface 501
screen-active-p 501
screen-logical-resolution 502
screen-internal-geometries 503
screen-monitor-geometries 504
screen-internal-geometry 505
screens 506
scroll 507
scroll-bar 509
scroll-if-not-visible-p 511
search-for-item 512
selection 512
selection-empty 514
set-application-interface 514
set-button-panel-enabled-items 516
set-clipboard 516
set-composition-placement 518
set-confirm-quit-flag 519
set-default-editor-pane-blink-rate 519
set-default-interface-prefix-suffix 520
set-default-use-native-input-method 522
set-display-pane-selection 522
set-drop-object-supported-formats 523
set-editor-parenthesis-colors 523
set-geometric-hint 525
set-hint-table 526
set-horizontal-scroll-parameters 527
set-interactive-break-gestures 528
set-list-panel-keyboard-search-reset-time 529
set-object-automatic-resize 530
set-pane-focus 535
set-rich-text-pane-character-format 535
set-rich-text-pane-paragraph-format 538
set-selection 540
set-printer-metrics 541
set-printer-options 542
set-text-input-pane-selection 544
set-top-level-interface-geometry 544
set-vertical-scroll-parameters 546
shell-pane 547
show-interface 548
show-pane 549
simple-layout 549
simple-network-pane 550
simple-pane 550
simple-pane-handle 561
simple-pane-visible-height 562
simple-pane-visible-size 562
simple-pane-visible-width 563
simple-pinboard-layout 564
simple-print-port 565
slider 566
sort-object-items-by 569
sorted-object 570
sorted-object-sort-by 570
sorted-object-sorted-by 571
start-gc-monitor 572
static-layout 573
stop-gc-monitor 574
stop-sound 575
switchable-layout 575
switchable-layout-switchable-children 577
tab-layout 577
tab-layout-panes 582
tab-layout-visible-child 582
text-input-choice 583
text-input-pane 584
uninstall-postscript-printer 654
unmap-typeout 655
update-all-interface-titles 655
update-interface-title 656
update-pinboard-object 656
update-screen-interface-titles 657
*update-screen-interfaces-hooks* 657
update-toolbar 658
virtual-screen-geometry 658
with-atomic-redisplay 659
with-busy-interface 660
with-dialog-results 661
with-document-pages 663
with-external-metafile 664
with-geometry 667
with-internal-metafile 669
with-output-to-printer 671
with-page 672
with-page-transform 673
with-print-job 673
with-random-typeout 674
wrap-text 675
wrap-text-for-pane 676
x-y-adjustable-layout 677

2  GP Reference Entries 679

2pi 679
analyze-external-image 680
apply-rotation 680
apply-rotation-around-point 681
apply-scale 682
apply-translation 683
augment-font-description 683
clear-external-image-conversions 684
clear-graphics-port 685
clear-graphics-port-state 685
clear-rectangle 686
compress-external-image 687
compute-char-extents 687
inset-rectangle 758
inside-rectangle 759
invalidate-rectangle 760
invert-transform 761
list-all-font-names 761
list-known-image-formats 762
load-icon-image 763
load-image 765
make-dither 767
make-font-description 768
make-graphics-state 769
make-image 769
make-image-access 770
make-image-from-port 771
make-sub-image 772
make-transform 773
merge-font-descriptions 774
offset-rectangle 775
ordered-rectangle-union 775
pi-by-2 776
pixblt 777
pixmap-port 778
port-drawing-mode-quality-p 778
port-graphics-state 779
port-height 780
port-string-height 780
port-string-width 781
port-width 781
postmultiply-transforms 782
premultiply-transforms 782
read-and-convert-external-image 783
read-external-image 784
rectangle-bind 785
rectangle-bottom 786
rectangle-height 786
rectangle-left 787
rectangle-right 787
rectangle-top 788
rectangle-union 788
rectangle-width 789
rect-bind 790
register-image-load-function 790
register-image-translation 791
reset-image-translation-table 792
separation 793
set-default-image-load-function 793
set-graphics-port-coordinates 794
set-graphics-state 795
transform 795
transform-area 796
transform-distance 796
transform-distances 797
transform-is-rotated 797
transform-point 798
transform-points 798
transform-rect 799
undefine-font-alias 800
union-rectangle 800
*unit-transform* 801
unit-transform-p 801
unless-empty-rect-bind 802
untransform-distance 802
untransform-distances 803
untransform-point 803
untransform-points 804
validate-rectangle 805
with-dither 806
with-graphics-mask 806
with-graphics-post-translation 808
with-graphics-rotation 809
with-graphics-scale 810
with-graphics-state 810
with-graphics-transform 812
with-graphics-transform-reset 813
with-graphics-translation 814
with-inverse-graphics 815
without-relative-drawing 815
with-pixmap-graphics-port 816
with-transformed-area 817
with-transformed-point 818
with-transformed-points 819
with-transformed-rect 819
write-external-image 820

3 COLOR Reference Entries 823

apropos-color-alias-names 823
apropos-color-names 824
apropos-color-spec-names 825
color-alpha 826
color-<component> 826
*color-database* 827
color-level 828
color-model 829
color-with-alpha 829
colors= 830
convert-color 831
define-color-alias 832
define-color-models 833
delete-color-translation 834
ensure-<command> 835
get-all-color-names 836
get-color-alias-translation 837
get-color-spec 838
load-color-database 839
make-gray 839
make-hsv 840
make-rgb 841
read-color-db 842
unconvert-color 843

Index 845
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 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 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 CAPI User Guide and the CAPI Reference Manual describe the CAPI. This is a library of classes, functions, and macros for developing graphical user interfaces for your applications. The CAPI User Guide is a tutorial guide to the CAPI, and the 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

\textbf{abort-dialog} \hfill \textit{Function}

\textbf{Summary} \hfill The \texttt{abort-dialog} function aborts the current dialog.

\textbf{Package} \hfill capi

\textbf{Signature} \hfill \texttt{abort-dialog \&rest ignored-args}

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

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

\begin{verbatim}
(unless (capi:abort-dialog)
  (foo))
\end{verbatim}

Above, \texttt{foo} will be called only if there is no current dialog.

It is not useful to do either:

\begin{verbatim}
(when (capi:abort-dialog)
  (foo))
\end{verbatim}

or

\begin{verbatim}
(progn
  (capi:abort-dialog)
  (foo))
\end{verbatim}

as in both cases it is not well-defined whether \texttt{foo} will be called if there is a current dialog.
Example

```lisp
(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-confirmert
interface

abort-exit-confirmert

Function

Summary Aborts the exiting of a dialog.

Package capi

Signature abort-exit-confirmert

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

If abort-exit-confirmert is called outside the exiting of a confirmert, 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-confirm
(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-confirm)
     value))

See also popup-confirm

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

Function

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

Package
capi

Signature
activate-pane pane

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

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

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

(let 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.

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

See also pane-interface-copy-object

append-items

Generic Function

Summary
Adds to the items in a collection.

Signature
append-items collection new-items

Arguments
collection A collection.

new-items A sequence.

Description
The generic function append-items adds the items in new-items to the collection collection.
This is logically equivalent to recalculating the collection items and calling (setf collection-items). However, append-items is more efficient and causes less flickering on screen.

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

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

apply-in-pane-process

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.

3. If the pane’s process is no longer active then apply-in-pane-process applies function directly.
4. **apply-in-pane-process-if-alive** is another way to call function in the CAPI process appropriate for pane. However it only does this if pane is alive so in particular, if pane does not have a process, it does not call function.

**Example**

Editor commands must be called in the correct process:

```lisp
(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")
```

**apply-in-pane-process-if-alive**

**Function**

**Summary**

Applies a function in the process associated with a pane.

**Package**

capi

**Signature**

apply-in-pane-process-if-alive pane function &rest args => nil

**Description**

The function **apply-in-pane-process-if-alive** applies function to args in the process that is associated with pane, if pane is alive.

This is like **apply-in-pane-process** except that function is called only if the pane is alive in the sense defined for the interface in **execute-with-interface-if-alive**. If pane does not have a process, then function is not called.

**See also**

apply-in-pane-process
execute-with-interface-if-alive
**arrow-pinboard-object**

**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 *graphics-state*. 
Example

Example

Example

Example

Example

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 func-
tion 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.

Notes
This function is implemented only in LispWorks for Win-
dows. 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.

Notes This function is implemented only in LispWorks for Windows. Load the functionality by (require "embed").

See also detach-sink ole-control-pane

beep-pane

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
browser-pane

**Class**

**Summary**
Embeds a pane that can display HTML. Implemented only on Microsoft Windows and Cocoa.

**Superclasses**
simple-pane

**Subclasses**
None

**Initargs**

: **before-navigate-callback**
A function that is called before navigating, or nil.

: **navigate-complete-callback**
A function that is called when navigation completes, or nil.

: **new-window-callback**
A function that is called before opening a new window, or nil.

: **status-text-change-callback**
A function that is called when there is a new status text or nil.

: **document-complete-callback**
A function that is called when a document is complete, or nil.

: **title-change-callback**
A function that is called when the title changes, or nil.

: **update-commands-callback**
A function that is called when the enabled status of commands related to the pane may need to change, or nil.
:internet-explorer-callback
Microsoft Windows specific: A function that is whenever there is an event from the underlying IWebBrowser2, or nil.

:navigate-error-callback
A function that is called when the pane fails to navigate, or nil.

:debug
A boolean specifying whether debugging mode is on or not.

:url
A string specifying the initial URL.

Accessors

browser-pane-navigate-complete-callback
browser-pane-new-window-callback
browser-pane-status-text-change-callback
browser-pane-document-complete-callback
browser-pane-title-change-callback
browser-pane-update-commands-callback
browser-pane-internet-explorer-callback
browser-pane-before-navigate-callback
browser-pane-navigate-error-callback
browser-pane-debug

Readers

browser-pane-url
browser-pane-successful-p
browser-pane-title

Description

A browser-pane is a pane that embeds a pane that can display HTML. Navigation in the pane happens either by the user clicking on hyperlinks, or by the application using browser-pane-navigate. The various callbacks gives the program information on what happens in the window and can be used to control (for example, to block or redirect pages).

browser-pane is implemented only on Microsoft Windows (where it embeds an IWebBrowser2) and Cocoa (where it uses WebKit).
The initarg :url specifies the initial URL. After being created, the pane automatically navigates to this URL.

When before-navigate-callback is non-nil, it is called before any navigation (whether programmatic or by the user), and gives the application control over whether to perform the navigation. The callback must have this signature:

```
before-navigate-callback pane url &key hyper-link-p sub-frame-p frame-name post-data headers &allow-other-keys => do-it-p
```

```
before-navigate-callback pane url &key sub-frame-p frame-name &allow-other-keys => do-it
```

`pane` is the pane that navigates, and `url` is a string to which it wants to navigate. `sub-frame-p` is true when the navigation is for a sub-frame inside the current URL, otherwise `sub-frame-p` is `nil`. `frame-name` is either `nil` or the name of a sub-frame when the navigation is to a sub-frame.

If before-navigate-callback returns `nil`, the navigation is cancelled.

**Note:** To perform a redirection, just call `browser-pane-navigate` to the required URL, and return `nil` from before-navigate-callback.

If new-window-callback is non-nil, it is called before the pane tries to open a new window. It must have this signature:

```
new-window-callback pane url &key context flags &allow-other-keys => do-it-p
```

`pane` is the pane that wants to open a new window, and `url` is a string containing the URL that the new window will navigate to. `context` is a string containing the URL of the page from which the request comes.

`flags` is implementation specific flags. On Cocoa `flags` is always 0. On Microsoft Windows `flags` contains bits from the NWMF enumeration.
If `new-window-callback` returns `nil`, the opening of the new window is cancelled. If `new-window-callback` returns `t` or is not supplied, it launches a browser using the OS settings.

On Microsoft Windows, `new-window-callback` is invoked from the "NewWindow3" event (or "NewWindow2" for old versions) of the sink of the underlying IWebBrowser2. If not cancelled, the pane opens a new normal Internet Explorer window.

If `document-complete-callback` is non-nil, it is called when the new document in the pane is complete. It must be a function with signature:

```lisp
document-complete-callback => pane url title
```

`url` is the loaded URL, and may be `nil` in the case of failure. `title` is a string that is associated with the URL `url` (or the previous URL if the latest call failed).

`document-complete-callback` is called when, as far as the system is concerned, all the data for the URL has been loaded and is displayed in the pane. There is only one call to `document-complete-callback` for each navigation of the pane.

If `navigate-complete-callback` is non-nil, it is called whenever a navigation completes. `navigate-complete-callback` can be called several times for each navigation of the pane. It must be a function with the signature:

```lisp
navigate-complete-callback pane url sub-frame-p =>
```

`pane` is the pane that is navigated. `url` is a string to which it navigated, unless the navigation failed, in which case `url` is `nil`. `sub-frame-p` is true when the navigation was in a sub-frame.

**Notes:** For most purposes the `document-complete-callback` is more useful than `navigate-complete-callback`. When `navigate-complete-callback` gets a `nil url`, the value of the URL in the
pane (that is, what the accessor \texttt{browser-pane-url} returns) is still set to the actual URL. The success flag (which you can read with \texttt{browser-pane-successful-p}) is set to \texttt{nil}.

\texttt{url} can be non-nil even if there was an error in the navigation, if the server supplied another URL. In this case, on Microsoft Windows only, the success flag is set to \texttt{redirected}. You can read it with \texttt{browser-pane-successful-p}.

If \texttt{navigate-error-callback} is non-nil, it is called when navigation fails for some reason. It should have this signature:

\begin{verbatim}
navigate-error-callback pane url &key http-code error-symbol implementation-error-code message frame-name sub-frame-p fatal &allow-other-keys => cancel
\end{verbatim}

\texttt{pane} is the navigating pane, and \texttt{url} is the URL that got the error.

If the failure is server-side failure, then \texttt{http-code} contains the http-code in the response of the server, otherwise (that is, when it failed to connect to a server) it is \texttt{nil}.

\texttt{error-symbol} is a keyword uniquely identifying the error. For an http error it is of the form \texttt{:HTTP_STATUS*}, and for requests with bad syntax \texttt{error-symbol} is \texttt{:bad-request}.

On Microsoft Windows \texttt{implementation-error-code} is the code in the "NavigateError" event. If \texttt{http-code} is non-nil then \texttt{implementation-error-code} and \texttt{http-code} will be the same. On Cocoa \texttt{implementation-error-code} will be the same as \texttt{http-code} in the case of server-side failure, otherwise it is one of the \texttt{NSURLError*} constants.

\texttt{fatal} is a boolean. A non-nil value means that nothing is going to be displayed in the pane to tell the user about the error.

\texttt{message} is a message saying what the error is. \texttt{sub-frame-p} is \texttt{t} when the navigation is for a sub-frame, otherwise \texttt{nil}. \texttt{frame-name} is the name of the frame.

The return value \texttt{cancel} of \texttt{navigate-error-callback} should be one of \texttt{nil}, \texttt{t}, or \texttt{:stop}, with these interpretations:
nil

On Microsoft Windows this means displaying either the substitution page from the server if there is one, or displaying automatically generated (by the underlying IWebBrowser2) error page.

t

Cancel. On Microsoft Windows this means not displaying the automatically generated error page, but displaying server substitution if there is any.

:stop

Stop the navigation immediately.

Note that the effect of the returned value cancel is only on the specific navigation, so it possible for a sub-frame to be stopped, while the main page and maybe other sub-frames complete.

On Cocoa there is no automatically generated error page, so the return value of cancel nil means the same as t, and both display whatever the server returned.

Note: To redirect on error, navigate-error-callback should just call browser-pane-navigate with the new page and return :stop.

If title-change-callback is non-nil, it is called when the title of the pane should change. It should have this signature:

title-change-callback pane new-title

new-title is a string, which the application should use as the title of the pane.

Note: In most cases, using the title argument of the document-complete-callback is more useful.

If status-text-change-callback is non-nil, it is called when the status text of the pane should change. It has this signature:

status-text-change-callback pane new-status-text

new-status-text is a string, which the application should use as the status text for the pane.
If `update-commands-callback` is non-nil, it is called when other panes (typically buttons or menu items) that are used to perform commands on the pane need to update. The callback has this signature:

```
update-commands-callback pane what enabled-p
```

Currently `what` can be one of:

- `:forward` Other panes that are used to go forward in the pane should be enabled or disabled.
- `:backward` Other panes that are used to go backward in the pane should be enabled or disabled.

Additionally on Microsoft Windows only, `what` can be:

- `t` Other panes that may try to anything with the pane may need updating. Note that this callback is called quite often with `what = t`, so make sure it usually does not do much work in this case.

`enabled-p` specifies whether the other panes should be enabled or disabled.

On Windows only, if `internet-explorer-callback` is non-nil, it is called for each event for the pane. It has the signature

```
internet-explorer-callback pane event-name args
```

`event-name` is a string specifying the event. `args` is a vector containing the arguments in order. The callback is called before any code that is used to implement the callbacks, which is called afterwards with the same argument vector. That means that the callback should not set anything in the vector, except when debugging.

`internet-explorer-callback` is intended to add functionality that is not given by the callbacks, and for debugging (but see also `:debug`). If you need more control, you probably wants to define your pane directly: for the basics see `examples/com/ole/html-pane.lisp`. 
debug specifies that the pane should be in debugging mode. Currently, on Microsoft Windows this means that it prints each event and the arguments that it receives. Whenever an event is sent to the sink associated with the embedded browser, the method name (which is the same as the event name in this case) and the argument are printed to mp:*background-standard-output*. On Cocoa it prints some diagnostics to mp:*background-standard-output*.

browse-pane-url returns the current url of the pane. Initially the value is the keyword :url, but once the browser completed navigation to some URL it is changed to this. Note that the url changes even if the navigation was not successful, as long as it was not stopped or cancelled and there was no substitution page.

browse-pane-title returns the title of the current document. Note that during navigation browse-pane-title and browse-pane-url may not be synchronised. They are synchronised when document-complete-callback is called, until the next before-navigate-callback call.

browser-pane-successful-p tests whether the navigation to the current URL completed successfully, returning nil for failure and t for success. On Microsoft Windows only it can also return :substituted, which means that the server returned an error but also supplied a substitution page. On Cocoa, browser-pane-successful-p returns only t or nil.

Notes

browser-pane and related APIs are implemented on Microsoft Windows and Cocoa only.

See also

browser-pane-navigate
browser-pane-busy
browser-pane-refresh
**Generic Functions**

**Summary**  
Controls a `browser-pane`.

**Signature**

- `browser-pane-navigate pane url => result`
- `browser-pane-busy pane => result`
- `browser-pane-go-back pane`
- `browser-pane-go-forward pane`
- `browser-pane-stop pane`
- `browser-pane-refresh pane &optional level`

**Arguments**

- `pane`  
  A `browser-pane`.
- `url`  
  A string.
- `level`  
  One of the keywords `:normal` and `:refresh_completely`.

**Values**

- `result`  
  A boolean.
- `name`  
  A string.

**Description**

These generic functions are used to control an instance of `browser-pane`.

`browser-pane-navigate` navigates to the supplied URL, that is it gets and displays the contents of the URL. Note that if there is any redirection, it is the redirected URL that is displayed.
browser-pane-navigate does the navigation asynchronously, so when the function returns the navigation has just started. If result is \texttt{t} then the navigation started, and if result is \texttt{nil} then some error in the url has already been detected. If the pane has an error callback, it already has been called in this case.

\textbf{Note:} \texttt{browser-pane-navigate} can be used to effect a redirection from inside the error before navigation and new-window callbacks.

\texttt{browser-pane-busy} tests whether the browser is currently navigating, returning true if it is.

\texttt{browser-pane-go-forward} and \texttt{browser-pane-go-back} navigate forward and back in the history, like the buttons on most web browsers.

\texttt{browser-pane-stop} stops the current navigation.

\texttt{browser-pane-refresh} refreshes the pane, which means re-reading the URL. \texttt{level} can be one of:

\begin{itemize}
\item \texttt{:normal} \hspace{1cm} Asks the server for the contents again. This is the default value of \texttt{level}.
\item \texttt{:refresh_completely} \hspace{1cm} Asks the server for the contents again without looking at any cache (it uses header Pragma:no-cache).
\end{itemize}

\textbf{Notes} \texttt{browser-pane} and related APIs are implemented on Microsoft Windows and Cocoa only.

\textbf{See also} \texttt{browser-pane}
**browser-pane-property-get**  
**browser-pane-property-put**  

*Generic Functions*

**Summary**  
Get or set value of a specified Windows property of the underlying browser.

**Signature**

- `browser-pane-property-get pane property-name`
- `browser-pane-property-put pane property-name value`

**Description**  
`property-name` has to be one of the properties listed in the Properties section of the documentation of IWebBrowser2 in the MSDN.

**Notes**

1. `browser-pane-property-get` and `browser-pane-property-put` are implemented on Microsoft Windows only.

2. `browser-pane-property-get` and `browser-pane-property-put` do not correspond to the methods "GetProperty" and "PutProperty" of IWebBrowser2.

**See also** `browser-pane`

**button**  

*Class*

**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 and Microsoft Windows:

:disabled-image The image for the button when disabled (or
    nil).

:selected-disabled-image The image used when the button is selected
    and disabled.

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

: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:

:selection-callback
   Called when the button is selected.

:callback
   For buttons this is a synonym of :selection-callback.

: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, including a .ico file on Microsoft Windows. 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
1. The simple-pane initarg foreground is not supported for buttons on Windows and Cocoa.
2. The disabled-image, armed-image and selected-disabled-image will work on Microsoft Windows provided you are running with the themed look-and-feel (which is the default). See "Using Windows themes" in the CAPI User Guide.

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

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

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

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

In the next example a button with an image instead of text is created.
(setq button
  (capi:contain
    (make-instance
      'capi:push-button
      :image
      (merge-pathnames
        "capi/applications/images/info.bmp"
        (sys:lispworks-dir "examples")))))

The following examples illustrate mnemonics:

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

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

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

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

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

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

See also
  button-panel
  callbacks
**button-panel**

*Class*

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

**Package**
capi

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

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

**Initargs**
:layout-class The type of layout for the buttons.
:layout-args Initialization arguments for the layout.
:callbacks The selection callbacks for each button.
:button-class The class of the buttons.
:images A list.
:disabled-images A list.
:armed-images A list.
:selected-images A list.
:selected-disabled-images A list.
:help-keys A list.
:default-button Specifies the default button.
:cancel-button Specifies the cancel button.
The following initargs controlling mnemonics apply only on Microsoft Windows:

:mnemonics  A list specifying mnemonics for the buttons.

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

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

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

Accessors  pane-layout

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

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

The layout of the buttons is controlled by a layout of class layout-class (which defaults to row-layout) but this can be changed to be any other CAPI layout. When the layout is created, the list of initargs layout-args is passed to make-instance.
Each button uses the callbacks specified for the button panel itself, unless the argument `callbacks` is specified. `callbacks` should be a list (one element per button). Each element of `callbacks`, if non-nil, will be used as the selection callback of the corresponding button.

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

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

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

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

<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
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*.

*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/.
```
Summary
Calculates the internal constraints of a pane.

Package
capi

Signature
calculate-constraints pane

Arguments
pane A CAPI pane or layout.

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

When the pane does not scroll in the relevant dimension, all the geometry hints (:external-min-width, :visible-max-height and so on) override the values that are computed by calculate-constraints.

See “Width and Height Constraints” in the CAPI User Guide for description of internal and external constraints.

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.
%max-width% The maximum width of pane.
%min-height% The minimum height of pane.
The maximum height of pane.
(See with-geometry.)

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

See also calculate-layout
define-layout
get-constraints
element
layout
with-geometry

### calculate-layout

**Generic Function**

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

**Package**
capi

**Signature**
calculate-layout layout x y width height

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

To set the x, y, width and height of the layout, use the macro with-geometry which works in a similar way as with-slots.
See also
- get-constraints
- with-geometry
- interpret-description

callbacks

Class

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

Package
capi

Superclasses
capi-object

Subclasses
collection
  item
  menu-object

Initargs
- :callback-type The type of arguments for the callbacks.
- :selection-callback
  The callback for selecting an item.
- :extend-callback
  The callback for extending the selection.
- :retract-callback
  The callback for deselecting an item.
- :action-callback
  The callback for an action.
- :alternative-action-callback
  The callback for an alternative action in choice and its subclasses.
Accessors

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

Description

Each callback function can be one of the following:

- `function` Call the function.
- `list` Apply the head of the list to the tail.

`:redisplay-interface`

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

`:redisplay-menu-bar`

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

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

- `:collection-data`
  
  (collection data)

- `:data`
  
  (item-data)

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

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

- `:element` (element)

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

- `:element-item` (element item)

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

- `:item` (item)
:item-element (item element)

:interface (item interface)

:interface-item (interface item)

:interface (interface)

:full (item-data item interface)

:focus The pane with the current input focus.

:none ()

nil ()

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

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

In a choice, the alternative-action-callback is invoked by a gesture which is the action-callback gesture modified by the Shift key on Windows and GTK+, and modified by the Command key on Cocoa.

alternative-action-callback is applicable only to choice and its subclasses.

Apart from being invoked with a different gesture, the alternative-action-callback has exactly the same semantics as action-callback.

Examples examples/capi/choice/alternative-action-callback.lisp
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 CAPI User Guide. In other cases, take care to modify displayed CAPI interfaces only in their own process: execute-with-interface and apply-in-pane-process are useful for this.

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

Example
(setq editor (capi:contain
 (make-instance 'capi:editor-pane
 :text "abc")))
(capi:apply-in-pane-process
 editor 'capi:call-editor editor "End Of Buffer")

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

See also
apply-in-pane-process
editor-pane
execute-with-interface
can-use-metafile-p

Function

Summary
Queries whether metafiles can be used.

Package
capi

Signature
can-use-metafile-p &optional screen => result

Arguments
screen An object accepted by the function convert-to-screen.

Values
result A boolean.

Description
The function can-use-metafile-p is the predicate for whether the default library (if no argument is passed) or a specified screen (if an argument is passed) can use metafiles. If the argument screen is supplied, it is converted to a screen by convert-to-screen.

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

See also
convert-to-screen
default-library

capi-object

Class

Summary
The class capi-object is the superclass of all CAPI classes.

Package
capi

Superclasses
standard-class
Subclasses

- item
- callbacks
- element
- interface
- pinboard-object

Initargs

- :name
  The name of the object.
- :plist
  A property list for storing miscellaneous information.

Accessors

- capi-object-name
- capi-object-plist

Description

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

Example

```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`
1 CAPI Reference Entries

**Signature**

\[
\text{(setf capi-object-property) value object property}
\]

**Description**

All CAPI objects contain a property list, similar to the symbol `plist`. The recommended ways of setting properties are `capi-object-property` and \((\text{setf capi-object-property})\). 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: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 yourself, as the button panel provides functionality for handling groups of buttons. However, check-button can be used if you need to have more control over the button’s behavior.

Example
The following code creates a check button.

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

The button can be selected and deselected using this code.

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

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

The following code disables and enables the button.

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

```lisp
(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
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.

```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**

**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, except on Mac OS X.

: **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 for lists 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. Therefore when calling (setf choice-selection) you must pass an integer or nil if interaction is :single-selection, and you must pass a list of integers if interaction is :multiple-selection or :extended-selection. 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.

Notes
When calling (setf choice-selection) you must pass an integer or nil when interaction is :single-selection. You must pass a list for other values of interaction.

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

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

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

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

(capi:choice-selected-item choice)
The selection is changed using the following code.

(setf (capi:choice-selection choice) 1)
(capi:choice-selected-item choice)

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

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

choice-selected-item

Generic Function

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

Package
capi

Signature
choice-selected-item choice

Signature
(setf choice-selected-item) item choice

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

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

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

(setq list (capi:contain
            (make-instance 'capi:list-panel
                            :items '(a b c d e)
                            :selection 2)))
The following code line returns the selection of the list panel.

(capi:choice-selected-item list)

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

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

(capi:choice-selected-item list)

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

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

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

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

Change the test function:

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

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

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

See also choice
choice-selected-items
collection
choice-selected-item-p Function

Summary Checks if an item is currently selected in a choice.

Package capi

Signature choice-selected-item-p choice item

Description 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

(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

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

Package capi
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.

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)
```

Note that `interaction :multiple-selection` is not supported for lists on Mac OS X.
See also

choice
choice-selected-item
collection

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.

Examples

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*)

This example also demonstrates choice-update-item:

examples/capi/choice/alternative-action-callback.lisp

See also choice

clipboard

Function

Summary Returns the contents of the system clipboard.

Package capi

Signature clipboard 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 clipboard returns the contents of the system clipboard as a string, or nil if the clipboard is empty.

format controls what kind of object is read. 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.

:metafile The object is a metafile.
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 \texttt{:metafile}} is not supported on GTK+ or X11/Motif.

The Microsoft Windows clipboard is usually set by the user with the \texttt{Ctrl+C} and \texttt{Ctrl+X} gestures. Note that the LispWorks editor uses these gestures when in Windows emulation mode.

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

The Mac OS X clipboard is usually set by the user with the \texttt{Command+C} and \texttt{Command+X} gestures.

See also
\begin{itemize}
  \item \texttt{clipboard-empty}
  \item \texttt{draw-metafile}
  \item \texttt{draw-metafile-to-image}
  \item \texttt{free-image}
  \item \texttt{free-metafile}
  \item \texttt{image}
  \item \texttt{selection}
  \item \texttt{set-clipboard}
  \item \texttt{text-input-pane-paste}
\end{itemize}
**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`

---

**clone**

*Generic Function*

**Summary**
Creates a copy of a CAPI object.

**Package**
capi

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

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

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

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

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

See also `capi-object`

---

cocoa-default-application-interface

Class

Summary
The class supporting application menus and message processing for a Cocoa application.

Package `capi`

Superclasses `interface`

Initargs

- `:message-callback`
  A function or `nil`.

- `:application-menu`
  `nil`, a `menu`, or the name of a slot containing a `menu` in the application interface.

- `:dock-menu`
  `nil`, a `menu`, or a function designator.

Accessors

- `application-interface-message-callback`
- `application-interface-application-menu`
- `application-interface-dock-menu`

Description
The class `cocoa-default-application-interface` supports the application menu, application messages and other functionality for a Cocoa application.

All Cocoa applications in LispWorks for Macintosh have an application interface, which is a hidden interface that provides the following:
1. The application menu (the leftmost menu in the menu bar, named after the application). See application-menu below.

2. The menu bar items that are displayed when no other interfaces are on the screen. See menu-bar-items in interface and menu-bar in define-interface.

3. An optional Dock context menu. See dock-menu below.

4. Optional application message processing. See message-callback below.

5. Control over the lifecycle and display-state of the application as a whole.

If you wish to override the defaults, then you should first define a subclass of cocoa-default-application-interface with your changes. Then set a single instance of this subclass as the application interface by calling set-application-interface before any CAPI functions that make the screen object (such as convert-to-screen and display).

Do not call display with a subclass of cocoa-default-application-interface - the application interface does not have a window on the screen and should be created in addition to the visible interfaces in your application.

When non-nil, message-callback should be a function with signature

\[ \text{interface message \&rest args} \]

message-callback will be called for various application messages. The interface argument will be the application interface and the message argument will be a keyword. The message argument will be one of the following:

:open-file This message is invoked when the user double-clicks on a document associated with the application or drags a document into the application icon. The args contain the name of the file to open.
:finished-launching

This message is invoked just after the user has started the application and all other initialization has been done (including any :open-file message if applicable). You can use it to open a default document for example. There are no args.

application-menu controls the application’s main menu. If this is nil, then a minimal application menu will be made using the title of the application interface, otherwise it should be a menu containing the usual items or the name of a slot containing such a menu in the application interface. Note that the Quit item in the application-menu needs to call destroy on the interface, rather than call lw:quit.

dock-menu provides a menu for use by the Mac OS X Dock icon. If the value is nil (the default), then the standard menu is used. If 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
- `:others-hidden` Hide Others
- `:all-normal` Show All

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

Example

See these files in the `examples` subdirectory of the LispWorks library:

capi/applications/cocoa-application.lisp
capi/applications/cocoa-application-single-window.lisp
delivery/macos/multiple-window-application.lisp
delivery/macos/single-window-application.lisp

See also `set-application-interface`
cocoa-view-pane  
Class

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

Package  
capi

Superclasses  
simple-pane  
titled-object

Initargs  
:view-class  
A string naming the view class to use.

:init-function  
A function that initializes the view class.

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

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

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

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

- If view-class is specified, then it should be a string naming the Cocoa view class to allocate. Otherwise the class NSView is allocated.

- If init-function is not nil, then it should be a function which is called with of two arguments, the pane and a foreign pointer to the newly allocated Cocoa view object. The function should initialize the Cocoa view object in whatever way is required, including invoking the appropriate Objective-C initialization method, and return the initialized view. If init-function is nil then the Objective-C method init is called and the result is returned.
After the Cocoa view has been initialized, the function `cocoa-view-pane-view` can be used to retrieve it.

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


**Example**

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

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

See also `cocoa-view-pane-view`

---

**cocoa-view-pane-view**

*Function*

**Summary**

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

**Package**

capi

**Signature**

cocoa-view-pane-view pane => view

**Arguments**

`pane` A `cocoa-view-pane`.

**Values**

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

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

**Example**

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

**See also**

`cocoa-view-pane`

---

**collect-interfaces**

**Generic Function**

**Summary**

Finds all interfaces of a given class.

**Package**

`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.

:items-map-function
A function that maps a function over the items.

:accepts-focus-p
Specifies that the collection should accept input. The default value is t.

:help-key An object used for lookup of help.

Accessors
collection-items
collection-print-function
collection-test-function

Readers
collection-items-count-function
collection-items-get-function
collection-items-map-function
help-key

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

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

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

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

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

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

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

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

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

Example

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

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

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

**Description**
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).

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

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

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

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

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

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

Package capi

Superclasses screen

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

See also element-screen, mono-screen

column-layout

Class

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

Package capi

Superclasses grid-layout

Initargs

:ratios The size ratios between the layout’s children.

:adjust The horizontal adjustment for each child.

:gap The gap between each child.

:uniform-size-p

If t, each child in the column has the same height.
Accessors

- layout-ratios

Description

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

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

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

Compatibility note

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

Example

```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

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.

Notes
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

Summary
Quits the Lisp session, potentially after user confirmation.

Package
capi

Signature
confirm-quit application-name
<table>
<thead>
<tr>
<th>Arguments</th>
<th>application-name A string.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>The function <code>confirm-quit</code> calls <code>quit</code>, potentially after confirmation from the user.</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Notes</th>
<th>The LispWorks IDE uses <code>confirm-quit</code>.</th>
</tr>
</thead>
<tbody>
<tr>
<td>See also</td>
<td><code>set-confirm-quit-flag</code></td>
</tr>
</tbody>
</table>
confirm-yes-or-no

Function

Summary
The function confirm-yes-or-no pops up a dialog box 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
(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

confirmer-pane

Function

Summary
Returns the pane associated with a confirmer interface.

Package
capi
Signature  

confirmer-pane interface => pane

Arguments  

interface  

A confirmer interface displayed by popup-confirmer.

Values  

pane  

The pane argument passed to popup-confirmer.

Description  

The function confirmer-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  

Function

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.

ccontain 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
Signature  \texttt{convert-relative-position from to x y => to-x, to-y}

Arguments  
\begin{itemize}
  \item \textit{from} A pane, interface or screen.
  \item \textit{to} A pane, interface or screen.
  \item \textit{x} An integer.
  \item \textit{y} An integer.
\end{itemize}

Values  
\begin{itemize}
  \item \textit{to-x} An integer.
  \item \textit{to-y} An integer.
\end{itemize}

Description  The function \texttt{convert-relative-position} converts the position \(x,y\) in the coordinate system of \textit{from} to that of \textit{to}.

Example  See the example file \texttt{examples/capi/elements/convert-relative-position.lisp}.

See also  
\texttt{top-level-interface-geometry}
\texttt{with-geometry}

\textbf{convert-to-screen}  \hspace{1cm} \textit{Function}

Summary  The \texttt{convert-to-screen} function finds the appropriate screen or container for a CAPI object.

Package  \texttt{capi}

Signature  \texttt{convert-to-screen \&optional object => result}

Arguments  
\begin{itemize}
  \item \textit{object} A CAPI object, a plist, or \texttt{nil}.
\end{itemize}

Values  
\begin{itemize}
  \item \textit{result} A screen or a container.
This finds the appropriate screen or container for the CAPI object *object*.

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

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

*object* can be a keyword representing the CAPI library. This is equivalent to using the `:library` key in the plist case below.

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

- **:display** The value is an X Window System display string describing the X display and screen to use. The default value is derived from the `DISPLAY` environment variable or (on Motif) the `-display` command-line option, or (on GTK+) the `--display` command-line option. If neither is supplied, the default is to use the default screen on the local host.

- **:host** The name of the host to use for the X Window System display. This key is valid only if no `:display` key/value is supplied. The default value is the local host.

- **:server-number** The number of the display server to use for the X Window System display. This key is valid only if no `:display` key/value is supplied. The default value is 0.

- **:screen-number** The number of the screen to use for the X Window System display. This key is valid only if no `:display` key/value is supplied. The default value is the default screen of the display.
:application-class

The value is a string naming the application class used for X Window System resources. The default value is "Lispworks". When running a delivered LispWorks image, you should specify the :application-class key if you want to provide application-specific resources.

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

See element for the description of widget-name.

Example GTK+ resource files are in lib/6-1-0-0/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 keys is supported on Motif only. Other libraries ignore it.

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

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

Example
(capi:convert-to-screen)
See also document-frame
screen
count-collection-items

Generic Function

Summary
Returns the number of items in a collection.

Package
capi

Signature
count-collection-items collection &optional representation

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

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

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

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

(capi:count-collection-items list)

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

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

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

current-dialog-handle

Function

Summary
Returns the underlying handle of the current dialog.
Package 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.

(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.
The function `current-pointer-position` returns the current x,y position of the pointer on the screen of `relative-to`, which defaults to the current screen.

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

See also `interface` `screen`

### `current-popup` Function

**Summary**

Returns the current popup pane if there is one.

**Signature**

`current-popup => result`

**Values**

`result` A pane or `nil`.

**Description**

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

If the dialog was raised by an explicit call to `display-dialog` or `popup-confirm`, `current-popup` returns the first argument of `display-dialog` or `popup-confirm`. 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-confirm`
**current-printer**  
*Function*

Summary  
Returns the currently selected printer object.

Package  
capi

Signature  
current-printer &key interactive => printer

Arguments  
interative  
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 confirmer 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 the default library.

On Linux, FreeBSD and xw86/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.
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

Description
The macro define-interface is used to define subclasses of interface, which when created with make-instance has the specified panes, layouts and menus created automatically. The slots and superclasses are used to describe the slots
and superclasses of name as in the defclass macro, except
that if superclasses is non-nil it must include interface or a
subclass of it.

define-interface accepts the same options as defclass,
plus 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, accessors or documentation by passing the
appropriate defclass keyword as one of the optional argu-
ments in the description. Therefore, if you need to find a
pane within an interface instance, you can provide an acces-
sor, or simply use with-slots.

The :panes option is a list of pane descriptions of the follow-
ing form

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

where slot-name is a name for the slot, pane-class is the class of
the pane being included in the interface, and initargs are the
initialization arguments for the pane - the allowed forms are
described below.
The `:layouts` option is a list of layout descriptions of the following form

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

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

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

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

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

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

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

`initargs` are the initialization arguments for the menu.

The values given in `initargs` under `:panes`, `:layouts` and `:menus` can be lists of the form
(:initarg keyword-name)
(:initarg key-spec)
(:initarg key-spec initarg-value)

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

keyword-name := any keyword

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

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

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

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

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

:interface-variable

The name of the variable containing the interface.

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

Here are a couple of menu examples:

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

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

```lisp
(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:

```lisp
(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:

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

(capi:contain (make-instance 'init2))
```
Here we increment the indices passed in the interface’s :select initarg before passing them in the :selected-items initarg of pane foo:

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

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

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

See also
interface
layout
menu

define-layout

Macro

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

Package
capi

Signature
define-layout name superclasses slots &rest options

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

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

Translate the layout's child descriptions.

calculate-constraints

Calculate the constraints for the layout.

calculate-layout

Layout the children of the layout.

See also
interpret-description
calculate-constraints
calculate-layout
layout

define-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

```lisp
(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

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.

Implementations of the methods in the :coclass and :interfaces options should be defined using **com:define-com-method**, **com:define-dispinterface-method** or **com:com-object-dispinterface-invoker**.
Notes  
Define-OLE-Control-Component is implemented only in LispWorks for Windows. Load the functionality by (require "embed").

See also  
ole-control-component

destroy  
Generic Function

Summary  
Closes a window and calls the destroy-callback.

Package  
capi

Signature  
destroy interface

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

There is a complementary function quit-interface which calls the interface’s confirm-destroy-function to confirm that the destroy should be done, and it is advisable to always use this unless you want to make sure that the interface’s confirm-destroy-function is ignored.

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

Example  
(setq interface (capi:display (make-instance 'capi:interface :title "Test Interface" :destroy-callback #'(lambda (interface)  
(capi:display-message "Quitting ~S"  
interface))))
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.

**Notes**
This function is implemented only in LispWorks for Windows. Load the functionality by `(require "embed")`.

**See also**
attach-simple-sink
ole-control-pane
### detach-sink

**Function**

**Summary**
Detaches a previously-attached sink.

**Package**
capi

**Signature**
detach-sink sink pane interface-name

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

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

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

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

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

Attached sinks are automatically disconnected when the object is closed.

**Notes**
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

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

### Values

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>interface</code></td>
<td>A CAPI interface.</td>
</tr>
</tbody>
</table>

### Description

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

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

On Windows and Motif, if `process` is `t`, then `interface` runs in a newly-created process. If `process` is `nil`, `interface` runs in the current process. Otherwise `process` is expected to be a CAPI process, and `interface` runs in it. A CAPI process is a `mp:process` which was created by calling `display`. You can pass only a CAPI process as `process`, because it needs to handle messages using the LispWorks event loop. The default value of `process` is `t`. 
On Cocoa, all CAPI interfaces run in the Cocoa Event Loop process (which is the main thread of LispWorks) and therefore the process argument is not supported. If the value of process is any process other than the Cocoa Event Loop process an error is signalled.

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

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

display returns its interface argument.

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

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

Example

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

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

**Summary**

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

**Package**

capi

**Signature**

display-dialog interface &key screen focus modal owner x y position-relative-to continuation 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 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 CAPI User Guide for details.

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

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

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

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

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

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

Notes
1. If you need to replace one dialog with another, you can use display-replacable-dialog and replace-dialog.

2. In a modal dialog at least one button which aborts or exits the dialog must be provided in interface. This is the programmer’s responsibility, as without such a button there is no way to clear the modal dialog. A straightforward way to add these buttons is to display the window via popup-confirm which adds the buttons for you.

Example

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

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

See also
abort-dialog
display
display-replacable-dialog
exit-dialog
interface
display-errors

Macro

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

Function

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 (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 editor-pane
text-input-pane
title-pane

display-pane-selected-text

Function

Summary Returns the selected text in a display-pane.

Package capi

Signature display-pane-selected-text display-pane => result

Arguments display-pane An instance of display-pane or a subclass.

Values result A string or nil.

Description The function display-pane-selected-text returns the selected text in display-pane, or nil if there is no selection.

See also display-pane
display-pane-selection-p
display-pane-selection

display-pane-selection

Function

Summary Returns the bounds of the selection in a display-pane.

Package capi

Signature display-pane-selection pane => start, end

Arguments pane A display-pane.

Values start, end Non-negative integers.
The function `display-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-display-pane-selection` `display-pane` `display-pane-selected-text` `display-pane-selection-p`

### Function

**display-pane-selection-p**

**Summary**

Returns true if there is selected text in a `display-pane`.

**Package**

capi

**Signature**

`display-pane-selection-p` `pane` => `selectionp`

**Arguments**

`pane`  
A `display-pane`.

**Values**

`selectionp`  
A boolean.

**Description**

The function `display-pane-selection-p` returns `t` if there is a selected region in `pane` and `nil` otherwise.

See also `set-display-pane-selection` `display-pane` `display-pane-selected-text` `display-pane-selection`
**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* 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**

**Summary**

Displays tooltip help on an output pane.

**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.
- `y` The vertical coordinate of the tooltip position.
- `text` The help text.
1 CAPI Reference Entries

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

Notes
1. On GTK+ `display-tooltip` is implemented only for GTK+ versions 2.12 and later
2. On GTK+ the `:x` and `:y` arguments might not be handled.

Compatibility note
On GTK+ `display-tooltip` is not implemented in LispWorks 6.0.

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

docking-layout

Class

Summary
A class that implements docking of panes.

Package
capi

Superclasses
simple-layout

Initargs
:items
A list of pane specifications. The panes become the items in the layout.

:controller
A controller for the layout, which can make multiple docking-layouts work together.

:docking-test-function
A function controlling whether a pane can be docked in a docking-layout.

:docking-callback
A function called when a pane is docked or undocked.

:divider-p
A boolean allowing a visible edge around the layout.
:orientation One of :horizontal or :vertical.

Accessors
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 \texttt{docking-layout}.
The default value is \texttt{nil}.

\texttt{docking-layout-items} always returns the items as lists, with the cdr containing the options and values.

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

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

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

\texttt{orientation} specifies whether the items are laid out horizontally or vertically. The default value is \texttt{:horizontal}.

Example
See the file \texttt{examples/capi/layouts/docking-layout.lisp}

See also
\texttt{docking-layout-pane-docked-p}
\texttt{docking-layout-pane-visible-p}
**docking-layout-pane-docked-p**  
*Function*

**Package**  
capi

**Signature**  
docking-layout-pane-docked-p docking-layout pane &key anywhere => dockep

**Signature**  
(setf docking-layout-pane-docked-p) dockep docking-layout pane => dockep

**Arguments**  
docking-layout An instance of docking-layout or a subclass.

pane A pane.

anywhere A boolean.

**Values**  
dockedp A boolean.

**Description**  
The function `docking-layout-pane-docked-p` returns a boolean indicating whether `pane` is currently docked.

If `anywhere` is `t`, `dockep` is true if `pane` is docked in any member of the Docking Group of `docking-layout`. If `anywhere` is `nil`, `dockep` is true only if `pane` is docked in `docking-layout` itself. The default value of `anywhere` is `nil`.

`(setf docking-layout-pane-docked-p)` may be used to change the docking state of `pane` in `docking-layout` only when `pane` is dockable in the Docking Group of `docking-layout`.

**See also**  
docking-layout

---

**docking-layout-pane-visible-p**  
*Function*

**Package**  
capi

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

Arguments  
- \textit{docking-layout}  An instance of \textit{docking-layout} or a subclass.
- \textit{pane}  A pane.

Values  \textit{visiblep}  A boolean.

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

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

\textit{See also}  \textit{docking-layout}

\texttt{document-container}  \textit{Class}

Package  \texttt{capi}

Superclasses  \texttt{capi-object}

Readers  \texttt{screen-interfaces}

Description  The class of the container in a \texttt{document-frame}.

A document container has some screen-like functionality, responding to \texttt{screen-internal-geometry} and \texttt{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.
(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
   n'(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
Description

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

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

The **selection-callback**, **extend-callback** or **retract-callback** is called as appropriate when items are moved between the lists.

There is no **action-callback** for **double-list-panel**.

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

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

Example

```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**

**Summary**
Initiates a dragging operation

**Package**
capi

**Signature**
\[
\text{drag-pane-object \hspace{1em} pane \hspace{1em} value \&key \hspace{1em} string \hspace{1em} plist \hspace{1em} image-function} \hspace{1em} \text{operations} \Rightarrow \text{operation}
\]

**Arguments**
- \textit{pane} A pane
- \textit{value} An object to be dragged
- \textit{string} A string to be dragged or \texttt{nil}
- \textit{plist} A plist of formats and objects to be dragged
- \textit{image-function} A function or \texttt{nil}
- \textit{operations} A list of operation keywords allowed for the dragged objects

**Values**
- \textit{operation} One of the operation keywords

**Description**
The function \texttt{drag-pane-object} initiates a dragging operation from within the pane \textit{pane}. It can only be called from within the button :press or button :motion callbacks of the \textit{input-model} of an \textit{output-pane}.

The \textit{value}, \textit{string} and \textit{plist} arguments are combined to provide an object to be dragged in various formats.

\textit{value} can be any Lisp object (not necessarily a string) to make available for dropping into a pane within the local Lisp image.

\textit{string} can be a string representation of \textit{value} to make available, or \texttt{nil}. If \textit{string} is \texttt{nil} and \textit{value} is a string, then that will be made available as the string.
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.

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.

Notes

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

2. image-function is only called on Cocoa. There is no way to specify an image when dragging on Microsoft Windows.
3. If :image is supplied in plist, the dragging mechanism automatically frees the image object as if by free-image when it no longer needs it.

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

See also simple-pane

**draw-metafile**

Function

Summary
Draws a metafile to a pane.

Package
capi

Signature
draw-metafile pane metafile x y width height

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

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

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

The graphics-state parameters transform, mask and mask-transform affect how the metafile is drawn. The other graphics-state parameters are taken from the metafile.

Notes
1. draw-metafile is supported on GTK+ only where Cairo is supported (GTK+ 2.8 and later).
2. Metafiles look bad on GTK+, because they transform the image rather than the drawing.

3. **draw-metafile** is not implemented on X11/Motif.

**Examples**

- examples/capi/graphics/metafile.lisp
- examples/capi/graphics/metafile-rotation.lisp

**See also**

- can-use-metafile-p
- clipboard
- draw-metafile-to-image
- free-metafile
- with-internal-metafile

---

**draw-metafile-to-image**

**Function**

**Summary**

Draws a metafile as an image.

**Package**

capi

**Signature**

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.
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 constrained, 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 compo-
nent. The default value of **alpha** is **nil**.

**Notes**

1. **draw-metafile-to-image** is supported on GTK+ only
   where Cairo is supported (GTK+ 2.8 and later).

2. Metafiles look bad on GTK+, because they transform the
   image rather than the drawing.

3. **draw-metafile-to-image** is not implemented on
   X11/Motif.
See also

- clipboard
- draw-metafile
- free-metafile
- with-internal-metafile

**drawn-pinboard-object**

**Class**

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

**Package**
capi

**Superclasses**

- pinboard-object

**Initargs**

:display-callback

Called to display the object.

**Accessors**

- drawn-pinboard-object-display-callback

**Description**
The `display-callback` is called with the output pane to draw on, the `drawn-pinboard-object` itself, and the `x`, `y`, `width` and `height` of the object, and it is expected to redraw that section. The `display-callback` should not draw outside the object's bounds.

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. However, it should not draw outside the pinboard object’s bounds.

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

\texttt{drop-object-collection-index \textit{drop-object} => index, placement}

\begin{verbatim}
(setf (drop-object-collection-index \textit{drop-object}) (values new-index new-placement))
\end{verbatim}

Arguments

- \textit{drop-object}: A \textit{drop-object}, as passed to the \textit{drop-callback}.
- \textit{new-index}: An integer.
- \textit{new-placement}: One of \texttt{:above}, \texttt{:item} or \texttt{:below}.

Values

- \textit{index}: An integer.
- \textit{placement}: One of \texttt{:above}, \texttt{:item} or \texttt{:below}.

Description

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

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

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

Notes

\texttt{drop-object-collection-index} should only be called within a \textit{drop-callback}. It is not supported on X11/Motif. See \texttt{simple-pane} for information about drop callbacks.

Example

For an example illustrating the use of drag and drop in a \texttt{choice}, see
\begin{verbatim}
examples/capi/choice/drag-and-drop.lisp
\end{verbatim}

See also

\texttt{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.

Notes
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.

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

**drop-object-drop-effect**  
*Function*

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

Package  capi

Signature  
\[
\text{drop-object-drop-effect} \; \text{drop-object} \Rightarrow \text{effect}
\]

Signature  
\[
(\text{setf drop-object-drop-effect}) \; \text{effect} \; \text{drop-object} \Rightarrow \text{effect}
\]

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.

- **:none**  No dragging is possible.

Notes  *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
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.

Notes
1. When receiving an image (by calling drop-object-get-object with the :image format), the received image should also be freed when you finish with it. However, it will be freed automatically when the pane supplied to drop-object-get-object is destroyed, so normally you do not need to free it explicitly.

2. 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 and examples/capi/choice/list-panel-drag-images.lisp.
See also  
set-drop-object-supported-formats  
simple-pane  

\textbf{drop-object-pane-x}  
\textbf{drop-object-pane-y}  

\textit{Generic Functions}  

\textbf{Summary}  
Gets the coordinates in the pane that an object is being dropped over.  

\textbf{Package}  
capi  

\textbf{Signature}  
drop-object-pane-x \textit{drop-object} \Rightarrow \textit{x-coord}  
drop-object-pane-y \textit{drop-object} \Rightarrow \textit{y-coord}  

\textbf{Arguments}  
\textit{drop-object}  
A \textit{drop-object}, as passed to the \textit{drop-callback}.  

\textbf{Values}  
\textit{x-coord, y-coord}  
Integers.  

\textbf{Description}  
The accessor functions \textbf{drop-object-pane-x} and \textbf{drop-object-pane-y} return the \textit{x} and \textit{y} coordinates within the pane that the object is being dropped over. This information is only meaningful when the pane is an instance of \textit{output-pane} or one of its subclasses.  

\textbf{Notes}  
\textbf{drop-object-pane-x} and \textbf{drop-object-pane-y} should only be called within a \textit{drop-callback}. They are not supported on X11/Motif. See \textit{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: `drop-object-provides-format drop-object format => 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.

Notes: `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`

### *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
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 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**

: text A string or nil.

: enabled t, nil or :read-only.
:buffer-modes A list specifying the modes of the editor buffer.

:buffer-name The name of the editor buffer.

:change-callback
A function designator, or nil.

:before-input-callback
A function designator, or nil.

:after-input-callback
A function designator, or nil.

: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.

:composition-face
Changes the editor face that is used by editor-pane-default-composition-callback to display the composition string. The default value is :default.

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
editor-pane-composition-face
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 LispWorks IDE via Tools > Preferences... > Environment > Styles > Colors and Attributes, 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 replace this behavior by supplying `:input-model` when you call `make-instance` or by `(setf capi:output-pane-input-model)`, though this
has an effect only if called before the pane is displayed. 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`.

Editor panes support GNU Emacs keys on all platforms. Additionally on Microsoft Windows they support Windows editor keys, on GTK+ and 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 GTK+ and 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


2. 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.

3. 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`. 
4. To control whether the native input method is used to interpret keyboard input, you can supply the output-pane initarg :use-native-input-method or call set-default-use-native-input-method.

5. The default value of composition-callback (see output-pane) is editor-pane-default-composition-callback.

**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 6.1 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:
\begin{verbatim}
(capi:define-interface updating-editor ()
  ()
  (:panes
    (numbers capi:list-panel
      :items '(1 2 3)
      :selection-callback 'update-editor
      :callback-type :interface
      :visible-min-height '(:character 3))
    (editor capi:editor-pane
      :text
      "Select numbers in the list above."
      :visible-min-width
      (list :character 35))))

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

(capi:display (make-instance 'updating-editor))
\end{verbatim}

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. It then displays the pane:
(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
  output-pane
  set-default-use-native-input-method

**editor-pane-blink-rate**

*Generic Function*

**Summary**

Returns the cursor blinking rate for an editor pane.

**Package**

capi

**Signature**

definer-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

capi

---

**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 "-%sHere's some more text..."))

See also editor-pane

*editor-pane-composition-selected-range-face-plist* Variable

Summary Can modify the face of the default editor composition string.

Initial Value (:inverse-p t)

Description The variable *editor-pane-composition-selected-range-face-plist* is a plist that is used to modify the face of the composition string when :selected-range and :selection-needs-face are passed in the plist to editor-pane-default-composition-callback. The plist is merged into the plist that is passed into editor-pane-default-composition-callback, so keywords in it override the keywords in the face.

See also editor-pane-default-composition-callback

editor-pane-default-composition-callback Function

Summary The default composition callback of the editor. Composition here means composing input characters into other characters by an input method.

Signature editor-pane-default-composition-callback editor-pane what

Description The function editor-pane-default-composition-callback is the default composition-callback of editor-pane. It may also be called by your program.
When called with \texttt{what = :start}, \texttt{editor-pane-default-composition-callback} sets the composition placement in the editor by calling \texttt{set-composition-placement}, and also makes it move the composition window following the user's mouse cursor movement.

When called with \texttt{what = :end}, it stops the following of the mouse cursor.

When called with a list (which needs to be a plist), \texttt{editor-pane-default-composition-callback} checks if it contains a keyword/value pair for \texttt{:string-face-lists}, and if it does displays it in the editor temporarily (until the next call to it). See the entry for \texttt{output-pane} for the description of the value \texttt{string-face-lists}.

By default, \texttt{editor-pane-default-composition-callback} uses the faces that are supplied in \texttt{string-face-lists}, but if the plist contains \texttt{:selection-needs-face} and \texttt{:selected-range}, it displays the selected range with a different face, by merging \texttt{*editor-pane-composition-selected-range-face-plist*} into the given face of the selected range.

This can be overridden by setting the \texttt{composition-face} in the \texttt{editor-pane}, or the global \texttt{*editor-pane-default-composition-face*} if the \texttt{composition-face} of the pane is \texttt{:default}. If \texttt{composition-face} is a true value then the exact behavior depends on its type:

\textbf{A plist}  
This is appended to each face plist in the the \texttt{string-face-lists}. In other words, it provides default values for the attributes of the face.

\textbf{An editor:face}  
Overrides the supplied face completely.
A function or a symbol

For string-face-list, funcalls it with two arguments, the pane and the supplied face plist, and uses the result (which may be an editor:face or a face plist).

editor-pane-default-composition-callback is the default value of composition-callback for editor-pane. This can be overridden by passing :composition-callback or using output-pane-composition-callback (see entry for output-pane).

The user-supplied callback may call editor-pane-default-composition-callback to do the actual display, potentially after modifying the argument when it is a plist.

See also set-composition-placement

*editor-pane-default-composition-face*  Variable

Summary  The default composition face for editor-pane.

Initial Value  nil

Description  The variable *editor-pane-default-composition-face* gives the default composition face for all editor-panes where the composition-face is set to :default.

:default is the default value for composition-face, so normally setting this variable affects the composition-face of all editor-panes.

See editor-pane-default-composition-callback for a description of how it is used.

See also  editor-pane-default-composition-callback
**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**  
*Generic Function*

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

**Package**  
capi

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

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

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

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

**editor-pane-selected-text-p**  
*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.
Package  
capi

Superclasses  
capi-object

Subclasses  
simple-pane
menu

Initargs  
:p: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.
:initial-constraints
            Specifies constraints (geometry hints) that apply to the element during the creation of the element's interface, but not after the interface is displayed.

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.
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 \texttt{nil}). Note that an element should not be used in more than one place at a time.

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

\texttt{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 \texttt{accepts-focus-p} returns true and the \texttt{:accepts-focus-p} initarg value is \texttt{:force}. On Motif and Cocoa, a pane acts as a tabstop if and only if the function \texttt{accepts-focus-p} returns true.

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

\texttt{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. \texttt{tab-layout} and \texttt{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 \texttt{widget-name} for these.

The actual widget name is the result of a call to \texttt{cl:string}, except when \texttt{widget-name} is a symbol, in which case the symbol name is downcased to derive the widget name.

If \texttt{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 \texttt{application-class} is prepended followed by a dot.
Example GTK+ resource files are in \texttt{lib/6-1-0-0/examples/gtk/}

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

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

All elements accept \texttt{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 \texttt{nil} (meaning no maximum), but the hints can be specified to change these values. The possible values for these hints are as follows:

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

- \texttt{max} The maximum size of the hints.
- \texttt{min} The minimum size of the hints.
- \texttt{+} The sum of the hints.
- \texttt{-} 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:

(:character integer)
(character integer)
The size of integer characters.

(:string string)
(string string)
The size of string.

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

(symbol-value foo)
The size of the symbol-value of foo.

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

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

initial-constraints must be a plist of constraints, where the keywords are geometry hints as described above.

Notes

1. 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.

2. 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.
3. 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 initargs `:min-width`, `:max-width`, `:min-height`, and `:max-height` 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))

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

Here is a simple example that demonstrates the use of the `element-parent` accessor to place elements.
(setq pinboard (capi:contain
    (make-instance
        'capi:pinboard-layout
        :visible-min-width 520
        :visible-min-height 395))
    (setq object
        (make-instance
            'capi:image-pinboard-object
            :x 10 :y 10
            :image
            (sys:lispworks-file
                "examples/capi/graphics/Setup.bmp")
            :parent pinboard))
    (capi:apply-in-pane-process
        pinboard #'(setf capi:element-parent) nil object)
    (capi:apply-in-pane-process
        pinboard #'(setf capi:element-parent) pinboard object)

These final two examples illustrate the effect of initial-constraints.

Create a pane that starts at least 600 pixel high, but can be made shorter by the user:

(capi:contain
    (make-instance 'capi:output-pane
        :initial-constraints '(:visible-min-height 600)))

Compare with this, which creates a pane at least 600 pixels high but which cannot be made shorter.

(capi:contain
    (make-instance 'capi:output-pane
        :visible-min-height 600))

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**

*Generic 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.
See also | attach-interface-for-callback element
**element-screen**

*Function*

**Summary**

Returns the screen that an element is associated with.

**Package**
capi

**Signature**
element-screen element => screen

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

**See also**
element

**ellipse**

*Class*

**Summary**

A pinboard object that draws itself as an ellipse.

**Package**
capi

**Superclasses**
pinboard-object

**Accessors**
filled

**Initargs**
:filled A boolean.

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

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

**ensure-area-visible**

*Generic Function*

**Summary**

Ensures an area is visible in a scrollable pane.

**Package**
capi
Signature  

\texttt{ensure-area-visible self x y width height}

Arguments  

\textit{self} \hspace{1em} A simple-pane with internal scrolling.

\textit{x,y} \hspace{1em} The coordinates of the origin of the area to make visible.

\textit{width, height} \hspace{1em} The dimensions of the area to make visible

Description  

The generic function \texttt{ensure-area-visible} ensures that the area specified by \textit{x}, \textit{y}, \textit{width} and \textit{height}, or at least part of it, is visible.

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

\begin{description}
\item[ensure-interface-screen] \textit{Function}
\item[Summary] Ensures that a top level interface is displayed on a given screen.
\item[Package] \texttt{capi}
\item[Signature] \texttt{ensure-interface-screen self &key screen}
\item[Description] The function \texttt{ensure-interface-screen} ensures that the top level interface is displayed on the given \textit{screen} (or the default) if \texttt{display} is called later without a \textit{screen} argument.

This allows the querying of font and color information associated with a particular screen. It returns the screen that is used.
\item[See also] \texttt{screen}
\texttt{display}
\texttt{interface}
\end{description}
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.
4. `apply-in-pane-process` and `apply-in-pane-process-if-alive` are other ways to call a function in the appropriate CAPI process. They take panes of all classes, not merely `interface`.

**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`
- `apply-in-pane-process-if-alive`
- `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.
Notes

All accesses (reads as well as writes) on a CAPI interface and its sub-elements should be performed in the interface process. Using `execute-with-interface-if-alive` is one way of ensuring this.

See also

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

---

### exit-confirmert

**Function**

#### Summary

Called by the **OK** button on a dialog created with `popup-confirmert`.

#### Package

capi

#### Signature

`exit-confirmert &rest dummy-args`

#### Description

The function `exit-confirmert` is called by the **OK** button on a dialog created using `popup-confirmert`, 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-confirmert`.

#### Example

This example demonstrates the use of `exit-confirmert` 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-confirmert`.

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

See also

- `popup-confirmert`
- `display-dialog`
- `interface`
Function

exit-dialog

Summary
Exits the current dialog.

Package
capi

Signature
exit-dialog value

Description
The function exit-dialog 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

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 1. 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.
2. The delete item callback (see delete-item-callback in tree-view) is called in extended-selection-tree-view with the second argument being a list of the selected items, unless interaction is :single-selection, in which case it behaves the same as in tree-view.

See also tree-view

filtering-layout

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 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

```
(defvar *things* (list "Foo" "Bar" "Baz" 'car 'cdr))

(capi:define-interface my-interface ()
  ((things :reader my-things
    :initform *things*))
  (:panes
    (my-things-list-panel
capi:list-panel
    :reader my-interface-list-panel
    :items things
    :visible-min-height `(:character ,(length *things*))
    (my-filtering
capi:filtering-layout
    :change-callback 'update-my-interface
    :reader my-interface-filtering))
  (:layouts
    (a-layout
capi:column-layout
    '(my-filtering my-things-list-panel)))
  (:default-initargs :title "Filtering example")
)

(defun update-my-interface (my-interface)
  (let* ((things (my-things my-interface))
    (filtered-things
      (multiple-value-bind (regexp excludep)
        (capi:filtering-layout-match-object-and-exclude-p
          (my-interface-filtering my-interface) nil)
        (if regexp
          (loop for thing in things
            when (if (find-regexp-in-string regexp
              (string thing))
              (not excludep)
              excludep)
            collect thing)
          things))))
  (setf (capi:collection-items
    (my-interface-list-panel my-interface))
    filtered-things)))

See also filtering-layout-match-object-and-exclude-p
filtering-layout-match-object-and-exclude-p

Function

Summary
Returns filtering parameters for a filtering-layout.

Package
capi

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

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 corresponding to to.
If there is no such edge, find-graph-edge returns nil.

See also find-graph-node

find-graph-node

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`. 
There are many uses of `find-interface` in the LispWorks IDE development environment.

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 examples below demonstrate the use of grid layouts as an alternative 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 supported on GTK+ only where Cairo is supported (GTK+ 2.8 and later).

Notes free-metafile is not implemented on 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**

<table>
<thead>
<tr>
<th>Summary</th>
<th>Frees a loaded sound object on Microsoft Windows and Cocoa.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Package</td>
<td>capi</td>
</tr>
<tr>
<td>Signature</td>
<td>free-sound sound</td>
</tr>
<tr>
<td>Arguments</td>
<td>sound An array returned by load-sound.</td>
</tr>
<tr>
<td>Description</td>
<td>The function free-sound unloads (frees) the loaded sound object sound.</td>
</tr>
<tr>
<td>Notes</td>
<td>free-sound is not implemented on GTK+ and Motif.</td>
</tr>
</tbody>
</table>
| See also | load-sound  
read-sound-file                                       |

### get-collection-item  
**Generic Function**

<table>
<thead>
<tr>
<th>Summary</th>
<th>Returns the item at a specified position in a collection.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Package</td>
<td>capi</td>
</tr>
<tr>
<td>Signature</td>
<td>get-collection-item self index</td>
</tr>
<tr>
<td>Description</td>
<td>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.</td>
</tr>
</tbody>
</table>
| 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

generate-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  

<table>
<thead>
<tr>
<th>parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>The parameters are returned as multiple values, one for each key passed in <code>keys</code> and in the same order as the arguments.</td>
</tr>
</tbody>
</table>

Description  

Retrieves the specified parameters of the horizontal scroll bar of `self`, which should be a displayed instance of a subclass of `simple-pane` which does internal scrolling (such as `editor-pane`).

The valid `keys` are:

- `:min-range`  
  The minimum data coordinate.
- `:max-range`  
  The maximum data coordinate.
- `:slug-position`  
  The current scroll position.
- `:slug-size`  
  The length of the scroll bar slug.
- `:page-size`  
  The scroll page size.
- `:step-size`  
  The scroll step size.

Note: For the other pane classes, such as `list-panel`, the underlying widget decides what the scroll range and units are.

Example  

See the following CAPI example files:

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

See also  

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

---

**get-page-area**  

Summary  

Calculates the dimensions of suitable rectangles for use with `with-page-transform`. 
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.

See also

- `set-printer-metrics`
- `printer-metrics`
- `with-page-transform`

---

**get-scroll-position**

**Function**

<table>
<thead>
<tr>
<th>Summary</th>
<th>Returns the current scroll position of a pane such as <code>list-panel</code>, <code>display-pane</code> or <code>tree-view</code>.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Package</td>
<td><code>capi</code></td>
</tr>
<tr>
<td>Signature</td>
<td><code>get-scroll-position pane dimension =&gt; position</code></td>
</tr>
<tr>
<td>Arguments</td>
<td><code>pane</code> A pane with built-in scrolling.</td>
</tr>
<tr>
<td></td>
<td><code>dimension</code> A keyword, either <code>:horizontal</code> or <code>:vertical</code>.</td>
</tr>
<tr>
<td>Values</td>
<td><code>position</code> An integer or <code>nil</code>.</td>
</tr>
<tr>
<td>Description</td>
<td>The function <code>get-scroll-position</code> returns the scroll position of the pane <code>pane</code> in the given <code>dimension</code>. <code>pane</code> 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 <code>list-panel</code>, <code>display-pane</code> and <code>tree-view</code>. In general, the units in the returned value <code>position</code> are unspecified, but they can be passed to the generic function <code>scroll</code> with <code>operation :move</code> to restore the position. For a <code>list-panel</code>, the vertical units are items.</td>
</tr>
</tbody>
</table>
position is nil if pane is not displayed on the screen, for example if get-scroll-position is called after pane is destroyed.

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.
The scroll page size.

:step-size The scroll step size.

**Note:** For the other pane classes, such as `list-panel`, the underlying widget decides what the scroll range and units are.

**Example**

See the following CAPI example files:

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

**See also**

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

---

**graph-edge**

*Class*

**Summary**

The class of objects that represent edges in a graph.

**Package**

capi

**Superclasses**

`graph-object`

**Initargs**

:from The node where the edge starts.

:to The node where the edge ends.

**Accessors**

- `graph-edge-from`
- `graph-edge-to`

**Description**

The class of objects that represent edges in a `graph-pane`. `from` and `to` are the nodes that the edge connects.

**See also**

- `graph-pane`
**graph-node**

*Class*

**Summary**
The class of objects that represent nodes in a graph.

**Package**
capi

**Superclasses**
graph-object

**Readers**
graph-node-x  
graph-node-y  
graph-node-width  
graph-node-height  
graph-node-in-edges  
graph-node-out-edges

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

**See also**
graph-edge  
graph-pane

**graph-node-children**

*Generic Function*

**Summary**
Returns the children of a graph node.

**Package**
capi

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

**Arguments**
node A graph-node.

**Values**
result A list.
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

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.

graph-pane

Class

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

Package `capi`

Subclasses `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.

:layout-x-adjust
The adjust value for the x direction.

:layout-y-adjust
The adjust value for the y direction.

:node-pinboard-class
The class of pane to represent nodes.

:edge-pinboard-class
The class of pane to represent edges.

:node-pane-function
A function to return a pane for each node.

Accessors

graph-pane-layout-function
graph-pane-roots

Description

A graph pane calculates the items of the graph by calling the children-function on each of its roots, and then calling it again on each of the children recursively until no more children are found. The children-function gets called with an item of the graph and should return a list of the children of that item.

Each item is represented by a node in the graph.

The layout-function tells the graph pane how to lay out its nodes. It can be one these values:
Lay the graph out from the left to the right.
Lay the graph out from the top down.
Lay the graph out from the right to the left.
Lay the graph out from the bottom up.

`layout-x-adjust` and `layout-y-adjust` act on the underlying layout to decide where to place the nodes. The values should be a keyword or a list of the form `(keyword n)` where `n` is an integer. These values of `adjust` are interpreted as by `pane-adjusted-position`. `:top` is the default for `layout-y-adjust` and `:left` is the default for `layout-x-adjust`.

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

The `node-pane-function` is called to create a pane for each node, and by default it creates an instance of `node-pinboard-class`. It gets passed the graph pane and the item corresponding to the node, and should return an instance of a subclass of `simple-pane` or `pinboard-object`.

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

`graph-pane` is a subclass of `choice`, so for details of its selection handling, see `choice`.

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

**Notes**

The `output-pane` initarg `:drawing-mode` controls quality of drawing in a `graph-pane`, including anti-aliasing of any text displayed on Microsoft Windows and GTK+.
Compatibility note

In LispWorks 4.3 the double click gesture on a graph-pane node always calls the action-callback, and the user gesture to expand or collapse a node is to click on the circle drawn alongside the node.

In LispWorks 4.2 and previous versions, the double click gesture was used for expansion and contraction of nodes and the action-callback was not always called.

Example

(defun node-children (node)
  (when (< node 16)
    (list (* node 2)
      (1+ (* node 2)))))

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

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

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

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

This example demonstrates a different style of graph output with right-angle edges and parent nodes being adjusted towards the top instead of at the center.
This example demonstrates the use of 
:node-pinboard-class to specify that the nodes are drawn as push buttons.

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

See also  
item-pinboard-object  
line-pinboard-object  
output-pane

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.
### `graph-pane-delete-objects`  
**Signature**  
`graph-pane-delete-objects graph-pane objects`  
**Arguments**  
- `graph-pane`  
- `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`  
**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

Summary
Returns or sets the direction of a graph.

Package
capi

Signature
graph-pane-direction graph-pane => direction
(setf graph-pane-direction) direction graph-pane => direction

Arguments
graph-pane A graph-pane.

Values
direction One of :forwards or :backwards.

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

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

Example
(setf gp
  (make-instance 'capi:graph-pane
      :layout-function :top-down))
=>
#<CAPI:GRAPH-PANE [0 items] 20603294>

(setf (capi:graph-pane-direction gp)
  :backwards)
=>
NIL

(setf (capi:graph-pane-layout-function gp)
  =>
  :TOP-DOWN

See also
capi:graph-pane
graph-pane-edges  

**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

**Function**

**Summary**
Returns the graph object at a given position in a graph.

**Package**
capi

**Signature**
`graph-pane-object-at-position graph-pane x y => object`

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

**Values**
- `object` A `graph-object`, or `nil`.
- `x, y` Non-negative numbers.

**Description**
The function `graph-pane-object-at-position` returns the `graph-object` (either a `graph-edge` or a `graph-node`) at the coordinates `x, y` in the graph `graph-pane`.

If there is no `graph-object` at position `x,y` then `graph-pane-object-at-position` returns `nil`.

**See also**
`graph-pane`

### graph-pane-select-graph-nodes

**Generic Function**

**Summary**
Selects nodes in a graph according to a predicate.

**Package**
capi

**Signature**
`graph-pane-select-graph-nodes graph-pane predicate`

**Arguments**
- `graph-pane` A `graph-pane`.
- `predicate` A function of one argument with boolean result.
graph-pane-update-moved-objects

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 con-
nected to the moved objects. You can add non-primary meth-
ods 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
column-layout

**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` in of the children in the column. The size of the layout is controlled by the constraints on the rows and columns.

For `grid-layout` description is either a two dimensional array or a list in the order specified by `orientation` (which defaults to `:row`). In the case of a list, one of `columns` or `rows` can be supplied to specify the dimensions (the default is two columns). As well as panes, slot names and strings, description may contain the element `nil`, which is interpreted as a special dummy pane with suitable geometry for resizable gaps. This special interpretation of `nil` in the description is specific to `grid-layout` and its subclasses.

The `x-ratios` and `y-ratios` slots control the sizes of the elements in a grid layout in the following manner:

The elements of `x-ratios` (or `y-ratios`) control the size of each child relative to the others. If an element in `x-ratios` (or `y-ratios`) is `nil` the child is fixed at its minimum size. Otherwise the size is calculated as follows

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

where `ratio-sum` is the sum of the non-nil elements of `x-ratios` (or `y-ratios`) and `ratio` is the element of ratios corresponding to the child. If this ideal ratio size does not fit the maximum or minimum constraints on the child size, and the constraint means that changing the ratio size would not assist the sum of the child sizes fitting the total space available, then the child is fixed at its constrained size, the child is removed from the ratio calculation, and the calculation is performed again. If `x-ratios` (or `y-ratios`) has fewer elements than the number of children, 1 is used for each of the missing ratios. Leaving `x-ratios` (or `y-ratios`) `nil` causes all of the children to be the same size.
The positions of each pane in the layout can be specified using \texttt{x-adjust} and \texttt{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 \texttt{:bottom-extend}, the cell immediately above will be extended to fill both rows in that column.

If \texttt{has-title-column-p} is true, then the items in the description which correspond to the first column are treated specially:

\begin{itemize}
  \item A string \quad Equivalent to specifying \texttt{(:title \textcolor{red}{string})}
  \item A list of the form \texttt{(:title \textcolor{red}{string} \ . \ options)}
    \begin{itemize}
      \item Make a title using the given list as initargs.
      \begin{itemize}
        \item options is a plist of options, which can include the keys \texttt{:title-font, :title-args, :mnemonic} or \texttt{:mnemonic-escape}. See \texttt{titled-object} for how these are processed.
      \end{itemize}
    \end{itemize}
  \item A list of the form \texttt{(:mnemonic-title \textcolor{red}{string} \ . \ options)}
    \begin{itemize}
      \item Make a title using the given list as initargs.
      \begin{itemize}
        \item string can contain the mnemonic escape.
        \item options is a plist of options, which can include the keys \texttt{:title-font,}
      \end{itemize}
    \end{itemize}
\end{itemize}
Note: mnemonics are not supported on all platforms.

Example

```lisp
(capi:contain (make-instance
   'capi:grid-layout
   :description '(*1* *2* *3*
                  *4* *5* *6*
                  *7* *8* *9*)
   :columns 3))

(capi:contain (make-instance
   'capi:grid-layout
   :description (list "List:"
                 (make-instance
                   'capi:list-panel
                   :items '(1 2 3))
                 "Buttons:"
                 (make-instance
                   'capi:button-panel
                   :items '(1 2 3))))

(capi:contain (make-instance
   'capi:grid-layout
   :description (list "List:"
                 (make-instance
                   'capi:list-panel
                   :items '(1 2 3))
                 "Buttons:"
                 (make-instance
                   'capi:button-panel
                   :items '(1 2 3)))
   :x-adjust '(:right :left)
   :y-adjust '(:center :bottom))
```
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

**Function**

**Summary**  The function `hide-interface` hides the interface containing a specified pane.

**Package**  capi

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

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

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

**See also**  interface  
show-interface  
quit-interface

hide-pane

**Function**

**Summary**  Hides the specified pane.

**Package**  capi

**Signature**  `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.

**Package**  
capi

**Signature**  
highlight-pinboard-object pinboard object &key redisplay

**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**
:imange  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

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

(cd (sys:lispworks-dir "examples/capi/"))

(setf image
  (capi:contain
   (make-instance
    'capi:image-pinboard-object
     :image "applications/images/info.bmp")))

(capi:apply-in-pane-process
 (capi:element-parent image)
 #'(setf capi:image-pinboard-object-image
 "graphics/Setup.bmp" image))

(capi:apply-in-pane-process
 (capi:element-parent image)
 #'(setf capi:image-pinboard-object-image
 "applications/images/info.bmp" image))

(capi:contain
 (make-instance
  'capi:image-pinboard-object
   :image "graphics/Setup.bmp"
   :title "LispWorks Splashscreen"
   :title-adjust :right
   :title-position :bottom))

See also

pinboard-layout

Class

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:
install-postscript-printer

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.
Compatibility note

This class was named `interactive-stream` in LispWorks 3.2 but has been renamed to avoid confusion (this class is not a stream but a pane that contains a stream). The class `interactive-stream` and its accessors `interactive-stream-top-level-function` and `interactive-stream-stream` have been kept for compatibility but may be dropped in future versions of LispWorks.

Example

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

```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 output stream))
         (return)))))))
 :best-height 300
 :best-width 300)
```

See also collector-pane
interactive-pane-execute-command  

**Summary**

Simulates user entry of commands in an interactive-pane.

**Package**

capi

**Signature**

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

**Arguments**

- `interactive-pane`  
  An interactive-pane.

- `command`  
  A Lisp form.

- `command-modification-function`  
  A function or nil.

- `editp`  
  A generalized boolean.

**Description**

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

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

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

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

**See also**

- interactive-pane
- listener-pane-insert-value
### interface

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

**Package**
capi

**Superclasses**
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-tooltips
A boolean determining whether Tooltip Help is enabled.

:help-callback
A function called when a user gesture requests help.
:top-level-hook
  A function called around the top level event handler.

:external-border
  An integer or nil.

:initial-focus
  A pane, a symbol naming a pane, or nil.

:display-state
  One of the keywords :normal, :maximized, :iconic and :hidden.

:transparency
  A real number in the inclusive range [0,1], used on Cocoa, later versions of Microsoft Windows, and GTK+.

>window-styles
  A list of keywords, or nil.

:toolbar-items
  A list of items for the toolbar.

:toolbar-states
  A toolbar state plist.

:default-toolbar-states
  A toolbar state plist.

:pathname
  A pathname designator.

:drag-image
  nil, t or an image specifier (that is, a value acceptable as the id argument of load-image).
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`
- `interface-pathname`
- `interface-drag-image`

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:** `create-callback` should be used only for operations that must be done with the interface already created and cannot be done in `interface-display`. Otherwise they should be either done in `initialize-instance` or between your calls to `make-instance` and `display`. An operation that needs to run after the interface is created but just before displaying the interface as an ordinary window (typical cases are font queries and loading images) can be put in the `interface-display:before` method. An operation that needs to run just after displaying the interface as an ordinary window can be put in the `interface-display:after` method.

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
```
x and y are measured from the top-left of the screen rectangle representing the area of the primary monitor (the primary screen rectangle).

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 iconifyp
```

override-cursor, if non-nil, specifies a cursor that is used instead of the cursor of each pane inside the interface. The default value of override-cursor is `nil`. See below for an example of setting and unsetting the override cursor. override-cursor is not supported on Cocoa. override-cursor is ignored by text-input-pane on GTK+.

If message-area is true, then the interface is created with a message area at the bottom. The text of the message area can be accessed using the titled-object accessor titled-object-message. The default value of message-area is `nil`.

enable-pointer-documentation is a boolean controlling whether Pointer Documentation is enabled, on Motif. The default value is `t`. The actual action is done by the help-callback.
enable-tooltip 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.

eexternal-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. \texttt{transparency} should only be used for top-level interfaces.

\textit{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:

\texttt{:no-geometry-animation}

Cocoa: Programmatic changes to window geometry happen without animation.

\texttt{: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.

\texttt{:toolbox}

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

\texttt{:borderless}

Cocoa, Microsoft Windows, GTK+ and Motif: A window with no external decoration or frame.

\texttt{:internal-borderless}

Cocoa and Motif: Remove the default border between the window's edge and its contents.

Microsoft Windows: Remove the default border between the window's edge and its contents for dialogs.
 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.

pathname specifies the interface pathname. You can get and set this with the accessor interface-pathname. The pathname may be displayed in some way to the user, depending on the GUI library.

Currently, only Cocoa uses pathname, in two ways:

- It makes the interface display a drag image on the title bar (This is the same image that is set by interface-drag-image, and the drag-image takes precedence if it not nil). The user can drag from the drag image, and if there is no drag-callback or if the drag-callback returns :default it will drag the pathname as a one item in a :filenames-list. For information about drag-callback, see simple-pane's description of :drag-callback and simple-pane-drag-callback.

- The context menu (invoked by right-mouse-click) on the drag image or on the title raises a menu containing the components of the path. Selecting a component opens the Finder with it.

drag-image is currently only effective on Cocoa. A non-nil value specifies that the interface should have a drag image, which on Cocoa is a small image (16x16px) to the left of the window title.
When the user drags this image, if the interface has a drag-callback it is called and if this returns non-nil LispWorks performs drag-and-drop with the image. See simple-pane for details of the drag-callback.

It is possible to have the image for aesthetic purposes only by supplying drag-image and not specifying a drag-callback. When drag-callback is non-nil, it can dynamically decide whether to allow a dragging, or to disallow dragging (by returning nil).

The image specification can be an already converted image (made by load-image, convert-external-image, make-sub-image or make-image-from-port). The image will be freed automatically when the interface is destroyed or when drag-image is set by (setf interface-drag-image). Otherwise the system uses load-image to create a new image, which is also freed automatically.

The value t for drag-image is interpreted specially: it means display some image. If drag-image is set to t after an image has already been set, it just displays the previous image. This is useful if an image was displayed but then removed by (setf interface-drag-image) with nil. If there was no previous image, a default image is displayed.

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.

5. On Cocoa in the presence of a cocoa-default-application-interface, an interface with no menus of its own and with :auto-menus nil uses the menu bar from the application interface.

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.

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

Remove the override cursor.

(setf (capi:interface-override-cursor interface)
default)

This example illustrates `top-level-hook`. Evaluate this form and then get an error by the interrupt gesture in the editor pane. (For example, the interrupt gesture is `Meta+Control+C` on Motif and `Control+Break` on Microsoft Windows). Then select the Destroy Interface restart.
(capi:display
 (capi:make-container
  (make-instance
   'capi:editor-pane)
  :top-level-hook
  #'(lambda (func interface)
      (restart-case (funcall func)
       (nil ()
        :report
        (list "Destroy Interface ~a" interface)
        (capi:destroy interface))))))

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-document-modified-p
interface-toolbar-state
interface-customize-toolbar

interface-customize-toolbar

Function

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

Signature interface-customize-toolbar interface

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

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

See also `interface` `toolbar`

**interface-display**

*Generic Function*

**Summary**
The function called to display an interface on screen.

**Package**
capi

**Signature**
`interface-display interface`

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

**Description**
The generic function `interface-display` is called by `display` to display an interface on screen.

The primary method for `interface` actually does the work. You can add :before methods on your own interface classes for code that needs to be executed just before the interface appears, and :after methods for code that needs to be executed just after the interface appears.

`interface-display` is useful when you need to make changes to the interface which require it to be already be created. Font queries and loading images are typical cases.

**Notes**
1. `interface-display` is called in the process of `interface`.

245
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:

```lisp
(capi:define-interface my-tree ()
 ((favorite-color :initform :blue))
 (:panes
  (tree
capi:tree-view
  :roots ' (:red :blue :green)
  :print-function
  'string-capitalize))
 (:default-initargs
  :width 200
  :height 200))

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

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

See also

display
interface

interface-display-title

Function

Summary

Returns the interface title to use on screen.

Package
capi

Signature

interface-display-title interface => string
## Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>interface</code></td>
<td>A CAPI interface</td>
<td></td>
</tr>
</tbody>
</table>

## Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>string</code></td>
<td>A string</td>
<td></td>
</tr>
</tbody>
</table>

## Description

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

This is equivalent to:

```lisp
(capi:interface-extend-title
 interface
 (capi:interface-title  interface))
```

## See also

- `interface-extend-title`
- `set-default-interface-prefix-suffix`

### Interface-document-modified-p

#### Function

#### Summary

Gets and sets the document-modified flag in the interface.

#### Package

capi

#### Signature

```
interface-document-modified-p interface => value
(setf interface-document-modified-p) value interface
```

#### Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>interface</code></td>
<td>A CAPI interface</td>
<td>A CAPI interface</td>
</tr>
</tbody>
</table>

#### Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>value</code></td>
<td>A boolean</td>
<td></td>
</tr>
</tbody>
</table>

#### Description

The function `interface-document-modified-p` gets and sets the document-modified flag in the interface `interface`.

Currently this only has a visible effect on Cocoa, where an interface whose document is modified is flagged by adding a dark dot in the middle of its Close button (the red button at top-left of the window).
On other platforms the document-modified state is merely remembered.

See also interface

### 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.
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. This function is deprecated. Use `top-level-interface-geometry` instead.

**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.

This function is deprecated. Use `top-level-interface-geometry` instead.

See also `top-level-interface-geometry`
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.
:pc

Alt is interpreted as Alt on Microsoft Windows and can be used for shortcuts.

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

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

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

:mac

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

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

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

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

See the chapter “Emulation” in the LispWorks Editor User Guide for more detail about the different styles.

Notes

On Motif the code to implement accelerators and mnemonics clashes with the LispWorks meta key support. Therefore the keyboard must be configured so that none of the keysyms connected to mod1 (see xmodmap) are listed in the variable...
capi-motif-library:*meta-keysym-search-list*, which must be also be non-nil. Note also that Motif requires Alt to be on mod1.

See also editor-pane

interface-match-p

Generic Function

Summary
Determines whether an interface is suitable for displaying initargs.

Package
capi

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

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

Values
matchp A boolean

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

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

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

See also locate-interface
### interface-menu-groups

**Generic Function**

**Summary**

Used when an embedded document sets the `menu-bar-items` to its menus, on Windows.

**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 title of the menu.

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

**Notes**

`interface-menu-groups` 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 interface-reuse-p returns a true value if interface is suitable for reuse with initargs.

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

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

See also:
- interface-match-p
- locate-interface

---

Function: interface-toolbar-state

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

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

\begin{description}
\item[\texttt{interpret-description} \hfill \textit{Generic Function}]\end{description}

\textbf{Summary}  
Converts an abstract description of a layout’s children into a list of the children’s geometry objects.

\textbf{Package}  
capi

\textbf{Signature}  
\texttt{interpret-description} \texttt{layout} \texttt{description} \texttt{interface}

\textbf{Description}  
The generic function \texttt{interpret-description} translates an abstract description of the \textit{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.

\textbf{Example}  
See the examples in the directory \texttt{examples/capi/layouts/}.

\textbf{See also}  
\texttt{parse-layout-descriptor}  
\texttt{define-layout}  
\texttt{layout}  
\texttt{interface}
**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**
```
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

```lisp
(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-pane-interface-copy-object**

*Generic Function*

**Summary**
Determines what pane-interface-copy-object returns from a choice.
Signature

`item-pane-interface-copy-object item choice interface => object, string, plist`

Description

The generic function `item-pane-interface-copy-object` is used by the method of `pane-interface-copy-object` that specializes on `choice` to decide what to return.

If only one item is selected, the `pane-interface-copy-object` method for `choice` returns what `item-pane-interface-copy-object` returns for this item. In this case all three of the return values are used.

If multiple items are selected, `pane-interface-copy-object` applies `item-pane-interface-copy-object` to each one, and returns a list of the returned objects as the first value, and a concatenation of returned strings (separated by newlines) as the second value. The plist is ignored if there more than one element.

The default method returns the item and its print representation (using the `print-function` of the `choice`), and no third return value.

You can define your own methods for `item-pane-interface-copy-object`. This is useful to make `active-pane-copy` work properly for a `choice`, in cases where the actual items in the choice are not the objects that are displayed in the choice as far as the user is concerned. For example, you may have a structure

```lisp
(defstruct my-item
    real-object
    color)
```

To give different colors to different lines in a `list-panel`. In this case `pane-interface-copy-object` (and hence `active-pane-copy` when the `list-panel` is active) will return the `my-item` structure, while the user will expect the real object. This can be fixed by adding a method:
(defmethod item-pane-interface-copy-object
  ((item my-item) pane interface)
  (let ((real-object (my-item-real-object item)))
    (values real-object
      (print-a-real-object real-object)))))

See also pane-interface-copy-object
active-pane-copy

item-pinboard-object

Class

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

Package
capi

Superclasses
pinboard-object
item

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

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

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

See also image-pinboard-object
pinboard-layout
**labelled-arrow-pinboard-object**  
*Class*

**Package**  
capi

**Superclasses**  
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  
A valid color specification, as defined for the graphics-state parameter foreground.

:text-background  
A valid color specification, as defined for the graphics-state parameter foreground, or the keyword :background, or nil.

**Accessors**  
labelled-line-text-foreground  
labelled-line-text-background
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.

text-foreground defines the color of the label text.

text-background defines the background for the text, which is the color used to draw a filled rectangle in the area of the text before drawing the text. The value :background means use the background of the pinboard-layout of the object. The value nil means do not draw a background rectangle. The default value of text-background is :background.

Example

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

See also pinboard-layout

layout

Class

Summary

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

Package capi

Superclasses titled-object

simple-pane
Subclasses

- simple-layout
- grid-layout
- pinboard-layout

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,

(setf (capi:layout-description pinboard-layout)
      (cons object
            (capi:layout-description pinboard-layout)))

is equivalent to

(capi:manipulate-pinboard pinboard-layout object
   :add-bottom)

See also define-layout

manipulate-pinboard

line-pinboard-object

   Class

Summary   A subclass of pinboard-object which displays a line drawn between two corners of the area enclosed by the pinboard object.

Package capi

Superclasses pinboard-object

Subclasses arrow-pinboard-object

right-angle-line-pinboard-object
Initargs

: start-x  The x coordinate of the start of the line.
: start-y  The y coordinate of the start of the line.
: end-x    The x coordinate of the end of the line.
: end-y    The y coordinate of the end of the line.

Description

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

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

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

Example

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

See also

move-line
pinboard-layout

line-pinboard-object-coordinates

Function

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

Package  capi

Signature  line-pinboard-object-coordinates object => start-x, start-y, end-x, end-y

Arguments  object  A line-pinboard-object.
Values

- start-x: An integer.
- start-y: An integer.
- end-x: An integer.
- end-y: An integer.

Description

The function line-pinboard-object-coordinates returns the start and end coordinates of the line-pinboard-object object.

See also

move-line

list-panel

Summary

The class list-panel is a pane that can display a group of items and provides support for selecting items and performing actions on them. Each item may optionally have an image.

Package

capi

Superclasses

choice
simple-pane
sorted-object
titled-object

Subclasses

list-view
multi-column-list-panel

Initargs

:right-click-selection-behavior
A keyword or nil. Controls the behavior on a right mouse button click.

:color-function
A function designator or nil. Controls item text color on Microsoft Windows, Cocoa and GTK+.
:alternating-background
  A boolean influencing the use of alternating background color on Cocoa and GTK+.

:filter
  A boolean. The default value is nil.

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

:filter-automatic-p
  A boolean. The default value is t.

:filter-callback
  A function designator or the keyword :default, which is the default value.

:filter-change-callback-p
  A boolean.

:filter-short-menu-text
  A boolean. The default value is nil.

:filter-matches-title
  A string, t or nil.

:filter-help-string
  A string, t or nil.

:keyboard-search-callback
  A function that is used to search for an item when the user types ordinary characters.

Initargs for handling images:

: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.
: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.

Accessors

list-panel-right-click-selection-behavior
list-panel-keyboard-search-callback
list-panel-image-function
list-panel-state-image-function

Description

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

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

To scroll a list-panel, call scroll with scroll-operation :move.

mnemonic-title is interpreted as for menu.

right-click-selection-behavior can take the following values:

nil Corresponds to the behavior in LispWorks 4.4 and earlier. The data is not passed.

All non-nil values pass the clicked item as data to the pane menu:
:existing-or-clicked/restore/discard
If the clicked item is not already selected, make it be the entire selection while the menu is displayed. If the clicked item is already selected, do not change the selection. If the menu is cancelled, the original selection is restored. If the user chooses an item from the menu, the selection is not restored.

:temporary-selection
A synonym for :existing-or-clicked/restore/discard.

:existing-or-clicked/restore/restore
If the clicked item is not already selected, make it be the entire selection while the menu is displayed. If the clicked item is already selected, do not change the selection. If the user chooses an item from the menu and the item’s callback does not set the selection then the original selection is restored after the callback. If the callback sets the selection, then this selection remains. The original selection is restored if the user cancels the menu.

:temporary-restore
A synonym for :existing-or-clicked/restore/restore.

:clicked/restore/discard
Make the clicked item be the entire selection while the menu is displayed. If the menu is cancelled, the original selection is restored. If the user chooses an item from the menu, the selection is not restored.
:temporary-always
   A synonym for :clicked/restore/discard.

:clicked/restore/restore
   Make the clicked item be the entire selection while the menu is displayed. If the user chooses an item from the menu and the item's callback does not set the selection then the original selection is restored after the callback. If the callback sets the selection, then this selection remains. The original selection is restored if the user cancels the menu.

:existing-or-clicked/discard/discard
   If the clicked item is not already selected, make it be the entire selection while the menu is displayed. If the clicked item is already selected, do not change the selection. The original selection is never restored, regardless of whether the user chooses an item from the menu or cancels the menu.

:discard-selection
   A synonym for :existing-or-clicked/discard/discard.

:clicked/discard/discard
   Make the clicked item be the entire selection. The original selection is never restored, regardless of whether the user chooses an item from the menu or cancels the menu.

:discard-always
   A synonym for :clicked/discard/discard.
:*no-change*

Does not affect the selection, but the clicked item is nonetheless passed as the data.

The default value of `right-click-selection-behavior` is `:no-change`.

`color-function` allows you to control the text colors on Microsoft Windows, Cocoa and GTK+. If `color-function` is non-nil, then it is a function used to compute the text color of each item, with signature

```
color-function list-panel item state => result
```

When `alternating-background` is true, the list panel is drawn with alternating background on Cocoa. On GTK+ it provides a hint, which the theme can override. Experience suggests that theme may draw with alternating background even when `alternating-background` is false, but when it is true they tend to draw it always. The default value of `alternating-background` is `nil`.

`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.

keyboard-search-callback should be a function with signature:

keyboard-search-callback pane string position => index, last-match, last-match-reset-time

pane is the list-panel, string is a string to match and position is the item index from which the system thinks that the search should start.

string contains the character that the user typed, appended to the "last match", if there is one. There is a "last match" if the previous call to keyboard-search-callback returned it (see below).

index is an index in the collection-items to move to. Apart from an integer inside the items range of the list-panel, this can be nil, which means do nothing, or :no-change, which selects the current item.

last-match is a string that should be recorded as the "last match" (if it is not a string, the "last match" is reset). This is prepended to the character in the next call, if the character is typed before the "last match" is reset.

last-match-reset-time is the time to wait before resetting the "last match", in seconds. Once this time passes, the last match is reset to nil. If last-match-reset-time is nil, the default value (which defaults to 1) is used. This default value can be changed by set-list-panel-keyboard-search-reset-time.
You can simplify the implementation of `keyboard-search-callback` by using `list-panel-search-with-function`.

As a special case, passing `:keyboard-search-callback` `t` tells CAPI to use its own internal search mechanism in preference to the native one. That can be useful on GTK+, where the default is to use the native search mechanism (for GTK+ versions after 2.4).

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


On Microsoft Windows, the following symbols are also recognized. They map to view images: `:view-large-icons`, `:view-small-icons`, `:view-list`, `:view-details`, `:view-sort-name`, `:view-sort-size`, `:view-sort-date`, `:view-sort-
Also on Microsoft Windows, these symbols are recognized. They map to history images: :hist-back, :hist-forward, :hist-favorites, :hist-addtofavorites and :hist-viewtree.

An image object, as returned by load-image.

An image locator object

This allowing a single bitmap to be created which contains several button images side by side. See make-image-locator for more information. On Microsoft Windows, it also allows access to bitmaps stored as resources in a DLL.

An integer

This is a zero-based index into the list panel’s image lists. 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.

If image-lists is specified, it should be a plist containing the following keywords as keys. The corresponding values should be image-list objects.

:normal

Specifies an image-list object that contains the item images. The image-function should return a numeric index into this image-list.
:state Specifies an image-list object that contains the state images. The state-image-function should return a numeric index into this image-list.

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

Generic Function

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

Package capi
list-panel-enabled

**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`  

**Function**

**Summary**

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

**Signature**

```lisp
(list-panel-items-and-filter list-panel)
```

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

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

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

but is more efficient.

```lisp
(setf list-panel-items-and-filter)  
(setf list-panel-items-and-filter list-panel)
```

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-search-with-function

Function

Summary
Searches a list-panel.

Signature
list-panel-search-with-function list-panel function arg &key start-index wrap-around reset-time

Arguments
list-panel A list-panel.
function A function taking two arguments. The first is arg, the second is an item in list-panel.
arg Any Lisp object.
start-index An integer, default 0.
reset-time A real number. The default is an internal value which can be set by set-list-panel-keyboard-search-reset-time.
wrap-around A boolean, default t.

Description
The function list-panel-search-with-function searches list-panel using function. list-panel-search-with-function is intended to simplify the implementation of the keyboard-search-callback of list-panel.
**list-panel-search-with-function** searches **list-panel** for a match. It applies **function** to each item and **arg**, until **function** returns non-nil.

When **function** returns non-nil, **list-panel-search-with-function** returns three values: the index of the item, **arg**, and **reset-time**.

The search starts at **start-index** if supplied, and at 0 otherwise. When the search reaches the end of the list panel and it did not start from 0, it wraps around to the beginning, unless **wrap-around** is supplied as **nil**. The default value of **wrap-around** is **t**.

**Example**

```lisp
(defun string-equal-prefix (string item)
  (let* ((start 0)
         (len (length item))
         (end (+ start (length string))))
    (and (>= len end)
         (string-equal string item
                        :start2 start
                        :end2 end))))
```

```lisp
(capi:contain
 (make-instance
  'capi:list-panel :items '("ae" "af" "bb" "cc")
  :keyboard-search-callback '#(lambda (pane string position)
   (capi:list-panel-search-with-function pane
     'string-equal-prefix ; or 'string-not-greaterp string
     :start position
     :reset-time 1
     :wrap-around t))))
```

Pressing "a" slowly cycles between "ae" and "af". Running the same example with **string-not-greaterp** instead causes "a" to cycle around all of the items.

**See also**

- **list-panel**
- **set-list-panel-keyboard-search-reset-time**
**list-panel-unfiltered-items**

*Generic Function*

**Summary**

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

**Signature**

```lisp
list-panel-unfiltered-items list-panel
(setf list-panel-unfiltered-items) new-items list-panel
```

**Description**

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

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

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

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

**See also**

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

**list-view**

*Class*

**Summary**

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

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

**Package**
capi
Superclasses  list-panel

Initargs  

: **view**  Specifies which view the list view pane shows. The default is :icon.

: **subitem-function**  Returns additional information to be displayed in report view.

: **subitem-print-functions**  Used in report view to print the additional information.

: **image-function**  Returns an image for an item

: **state-image-function**  Returns a state image for an item.

: **image-lists**  A plist of keywords and image-list objects.

: **columns**  Defines the columns used in report view.

: **auto-reset-column-widths**  Determines whether columns automatically resize. Defaults to :all.

: **auto-arrange-icons**  Determines whether icons are automatically arranged to fit the size of the window.

: **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
- list-view-auto-arrange-icons
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. See also auto-arrange-icons, below.

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

- 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:

:title The column heading.
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.

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.

If auto-arrange-icons is true, then the icons are automatically arranged to fit the size of the window when the view is showing :icon or :small-icon. The default value of auto-arrange-icons is nil.

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.

Notes
1. list-view is not implemented on Cocoa.
2. 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* \(\Rightarrow\)  *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:

```lisp
(setq cur4 (capi:load-cursor "C:/Temp/Animated_Cursors/la.cur"))
```

In this extended example, firstly we load a custom cursor for two platforms:

```lisp
(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:

```lisp
(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:

```lisp
(capi:apply-in-pane-process oo
  (lambda ()
    (setf (capi:simple-pane-cursor oo) :default)))
```

And we can restore the custom cursor:

```lisp
(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 on Microsoft Windows and Cocoa.

**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`.

**Notes**
1. The array `sound` contains the contents of the file. Its bytes are interpreted by the OS functions, so the format can be whatever they can deal with, for example WAV on
Microsoft Windows. The fact that this date is represented as an `(unsigned-byte 8)` array in Lisp does not constrain the output size.

2. `load-sound` is not implemented on GTK+ and Motif.

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

**Function**

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

**Package**
capi

**Signature**
`lower-interface pane`

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

make-container

Generic Function

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

Package
capi

Signature
make-container element &rest interface-args

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

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

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

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

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

(capi:display (capi:make-container
    (make-instance 'capi:text-input-pane)))

See also
contain
display
interface
element

make-docking-layout-controller

Function

Package capi

Signature
make-docking-layout-controller => controller

Values
controller A docking layout controller.

Description
The function make-docking-layout-controller returns a
docking layout controller object for use as the controller ini-
targ in docking-layout.

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

See also docking-layout

make-foreign-owned-interface

Function

Summary Creates a dummy interface which allows another
application’s window to be the owner of a CAPI dialog.

Package capi

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

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>handle</td>
<td>A Microsoft Windows hwnd.</td>
</tr>
<tr>
<td>name</td>
<td>A string naming interface.</td>
</tr>
</tbody>
</table>

Values

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>interface</td>
<td>An instance of foreign-owned-interface.</td>
</tr>
</tbody>
</table>

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 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 an icon resource in a Windows 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 function make-icon-resource-image-set constructs an image set object that identifies an image stored as a icon resource in a DLL on Microsoft Windows.

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.

Notes
make-icon-resource-image-set is only available in Lisp-Works for Windows.

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

**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

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>pane</code></td>
<td>A pane.</td>
</tr>
<tr>
<td><code>items</code></td>
<td>A list of <code>menu-objects</code>.</td>
</tr>
<tr>
<td><code>title</code></td>
<td>A string or <code>nil</code>.</td>
</tr>
<tr>
<td><code>menu-name</code></td>
<td>A string or <code>nil</code>.</td>
</tr>
<tr>
<td><code>component-p</code></td>
<td>A boolean.</td>
</tr>
</tbody>
</table>

## Values

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

## 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**
- **pane**
  A pane in an interface.
- **interface**
  An interface or nil.
- **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 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.

```lisp
(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

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

Package
capi

Signature
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 function make-resource-image-set constructs an image set object that identifies an image stored as a bitmap resource in a DLL on Microsoft Windows.

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.

Notes
make-resource-image-set is only available in LispWorks for Windows.

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 on Windows.

**Package** capi

**Signature**

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

**Arguments**

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

**Values**

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

**Description**

The function `make-scaled-general-image-set` 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, and `image-count` specifies the number of sub-images in both images. That is, the sub-images are scaled to this size.

The default value of `image-count` is 1.

**Notes**

`make-scaled-general-image-set` is only available in Lisp-Works for Windows.

**See also**

`image-set`

`make-general-image-set`
Function

make-scaled-image-set

Summary
Creates an image set by scaling the images of another image set on Windows.

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 object.

Values

image-set An image-set object.

Description
The function make-scaled-image-set 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. It is unspecified what happens if image-count is different from the image count in base-image-set.

Notes
make-scaled-image-set is only available in LispWorks for Windows.

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. The default value of key is *identity*.
- **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  
```
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

**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** 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**.

**Notes**

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:

\[
\text{(capi:apply-in-pane-process pl } \\
\text{ (lambda (pp) } \\
\text{ (dotimes (y 15) } \\
\text{ (let ((po (capi:pinboard-object-at-position pp 10 (* y 30)))) } \\
\text{ (when po (capi:manipulate-pinboard pp po :delete)))))) pl)
\]

Remove all line-pinboard-objects:

\[
\text{(capi:apply-in-pane-process pl 'capi:manipulate-pinboard pl } \\
\text{ (lambda (x) } \\
\text{ (typep x 'capi:line-pinboard-object)) :delete-if)
\]

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

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

(capi:map-collection-items collection 'princ-to-string nil)
```

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

Description
The function map-pane-children applies function to pane’s immediate children.

If visible is true, then function is applied only to the visible children.

If test is non-nil, it is a function which is applied first to each child, and only those for which test returns a true value are then passed to function.
If `reverse` is non-nil, the order in which the children are processed is reversed.

**Example**

This example constructs a pinboard containing random ellipses. A repainting function is mapped over them, restricted to those with width greater than height.
(defun random-color ()
  (aref #:red :blue :green :yellow :cyan
       :magenta :pink :purple :black :white)
  (random 10)))

(defun random-origin ()
  (list (random 350) (random 250)))

(defun random-size ()
  (list (+ 10 (random 40))
        (+ 10 (random 40))))

(setf ellipses
  (capi:contain
   (make-instance 'capi:pinboard-layout
                  :children
               (loop for i below 40
                     for origin = (random-origin)
                     for size = (random-size)
                     collect
                   (make-instance 'capi:ellipse
                                  :x (first origin)
                                  :y (second origin)
                                  :width (first size)
                                  :height (second size)
                                  :graphics-args
                               (list :foreground
                                      (random-color))
                                  :filled t))))

(defun repaint (ellipse)
  (setf (capi:pinboard-object-graphics-args ellipse)
        (list :foreground (random-color))
        (capi:redraw-pinboard-object ellipse t))

(defun widep (ellipse)
  (capi:with-geometry ellipse
   (> capi:%width% capi:%height%))

(capi:map-pane-children ellipses 'repaint :test 'widep)

 See also  map-pane-descendant-children
### `map-pane-descendant-children`  
*Generic Function*

**Summary**
Calls a function on each of the descendant panes of a pane.

**Package**
capi

**Signature**

```
map-pane-descendant-children pane function &key visible test reverse leaf-only
```

**Arguments**

- **pane**
  A CAPI pane.

- **function**
  A function of one argument.

- **visible**
  A boolean. The default value is `nil`.

- **test**
  A function of one argument, or `nil`. The default is `nil`.

- **reverse**
  A boolean. The default value is `nil`.

- **leaf-only**
  A generalized boolean. The default value is `nil`.

**Description**

The function `map-pane-descendant-children` applies `function` to `pane`'s descendant panes (that is, the children and each of their children recursively), depth first.

If `visible` is true, then `function` is applied only to the visible descendant panes.

If `test` is non-nil, it is a function which is applied first to each descendant pane, 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.

If `leaf-only` is true, then `function` is applied only to those panes which do not have children.

**See also**

- `map-pane-children`
- `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

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.

Class
capi
element
titled-menu-object

Summary

Package capi

Superclasses element
titled-menu-object

Initargs

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.

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

\textbf{Note:} \textit{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 \textit{items-function}, if you supply it, should work at this early stage.

If an item is not of type \texttt{menu-object}, then it gets converted to a \texttt{menu-object} with the item as its data. This function is called before the \textit{popup-callback} and the \textit{enabled-function} which means that they can affect the new items.

To specify a mnemonic in the menu title, you can use the initarg :\texttt{mnemonic}. The value \texttt{mnemonic} can be:

- An integer: The index of the mnemonic in the title.
- A character: The mnemonic in the title.
- \texttt{nil}: A character is chosen from a list of common mnemonics, or the :\texttt{default} behavior is followed. This is the default.
- :\texttt{default}: A mnemonic is chosen using some rules.
- :\texttt{none}: The title has no mnemonic.

An alternative way to specify a mnemonic is to pass \texttt{mnemonic-title} (rather than \texttt{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 \texttt{mnemonic-title} by \texttt{mnemonic-escape}, and \texttt{mnemonic-escape} is removed from \texttt{mnemonic-title} before the text is displayed. For example:

:mnemonic-title "&Open File..."
At most one character can be specified as the mnemonic in \texttt{mnemonic-title}. To make \texttt{mnemonic-escape} itself appear in the button, precede it in \texttt{mnemonic-title} with \texttt{mnemonic-escape}. For example:

\texttt{:mnemonic-title "&Compile & Load File..."}

If \texttt{image-function} is non-nil, it should be a function of one argument. \texttt{image-function} is called with the data of each menu item and should return one of:

\texttt{nil} \hspace{1cm} No image is shown.

\texttt{An image object} \hspace{1cm} The menu displays this image.

\texttt{An image id or external-image} \hspace{1cm} The system converts the value to a temporary image for the menu item and frees it when it is no longer needed.

If \texttt{image-function} is \texttt{nil}, no items in the menu have images. This is the default value.

\begin{enumerate}
\item On Cocoa and GTK+, menu items can contain both images and strings, so the \texttt{print-function} should return the appropriate string or "" if no string is required. On Microsoft Windows and Motif, if there is an image then the string is ignored. You can test programmatically whether menus with images are supported with \texttt{pane-supports-menus-with-images}.
\item 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.
\item 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}.
\end{enumerate}
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
(defun display-menu (menu)
  (capi:contain (make-instance 'capi:menu
                              :title "Test"
                              :items '(:red :green :blue)))

(defun display-menu (menu)
  (capi:contain (make-instance 'capi:menu
                              :title "Test"
                              :items '(:red :green :blue)
                              :print-function 'string-capitalize))

(defun display-menu (menu)
  (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
(defun display-menu (menu)
  (setq submenu (make-instance 'capi:menu
                                :title "Submenu...
                                :items '((1 2 3)))))

(defun display-menu (menu)
  (capi:contain (make-instance 'capi:menu
                              :title "Test"
                              :items (list submenu)))

Here is an example showing how to use the `items-function`:

```lisp
(defun display-menu (menu)
  (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 "M&nemonic Title"
    :items '(1 2 3)))

(capi:contain (make-instance 'capi:menu
    :mnemonic-title "M&e && You"
    :items '("Me" "You")))
```

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

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

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

**menu-component**

**Class**

**Summary**
The class `menu-component` is a choice that is used to group menu items and submenus both visually and functionally. The items contained by the `menu-component` appear separated from other items, menus, or menu components, by separators.
Package: 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 value which is suitable for passing to the `choice` accessor (`setf choice-selection`). This will be `nil`, or a single index (for `interaction :single-selection`), or a list of item indices (for `interaction :multiple-selection` and `:extended-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
```

```lisp
```
(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.

:enabled-function-for-dialog
nil, t, :same-as-normal or a function designator. Determines enabled state when a dialog is on screen.

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 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

```
(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-\texttt{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.

**Notes**

The function *enabled-function* should not display a dialog or do anything that may cause the system to hang. In general this means interacting with anything outside the Lisp image, including files, databases and so on.

**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 on Windows.

Package capi
Signature  
merge-menu-bars frame document => menus

Arguments  
frame       A document-frame.
document    An interface or nil.

Values  
menus       A list of menu objects.

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:

(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.

Notes  
merge-menu-bars is implemented only in LispWorks for Windows.

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

**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 \textit{fill} is non-nil the editor fills each paragraph in the buffer. If \textit{fill} is a fixnum then the buffer is filled at that width. If \textit{fill} is :\texttt{default} (the default value) and \textit{fixed-fill} is supplied then the value \textit{fixed-fill} is used. Otherwise the buffer is filled to the window width.

\textit{fixed-fill} defaults to \texttt{nil}.

See also \texttt{editor-pane}

\textbf{mono-screen} \hspace{1cm} \textbf{Class}

Summary \hspace{1cm} The \texttt{mono-screen} class is created for monochrome screen.

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

Superclasses \hspace{1cm} \texttt{screen}

Description \hspace{1cm} This is a subclass of \texttt{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 \texttt{color-screen}

\textbf{move-line} \hspace{1cm} \textbf{Generic Function}

Summary \hspace{1cm} Moves a \texttt{line-pinboard-object}.

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

Signature \hspace{1cm} \texttt{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.
- **end-x**
  The x coordinate of the end of the line.
- **end-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 a fixed width of the column.

- **:default-width** Specifies the default initial width of the column. The user can resize it. If :width is supplied it overrides :default-width.

- **:visible-min-width** Minimum width of the column.

- **:gap** Specifies an additional gap alongside the text in the column. :gap is not supported consistently across platforms (see Notes below).

The values of :width, :visible-min-width and :gap are interpreted as standard geometric hints. See *element* for information about these hints.

*columns* should indicate how many columns to display. At a minimum the value needs to be (() ()) for two columns without any titles

*header-args* is a plist of initargs passed to the header which displays the titles of the columns. The header object is a collection. The following collection initargs are useful to pass in *header-args*:
:selection-callback
   The callback for clicking on the header.

:callback-type
   Defines the arguments of the selection-callback.

:items
   The items of the header object. Note that:items overrides :title if that is supplied in columns.

:print-function
   Controls how each of items is printed, providing the title of each column.

header-args may also contain the keyword :alignments. The value should be a list of alignment keywords, each of which is interpreted like an :adjust value in columns. The alignment is applied to the title only.

If auto-reset-column-widths is true, then the widths of the columns are recomputed when the items of the multi-column-list-panel are set.

Notes

1. Similar and enhanced functionality is provided by listview.

2. On Microsoft Windows, :width in a column specification does not actually make the column width be fixed, though it does supply the initial width.

3. On Microsoft Windows, :gap in a column specification adds the gap on both sides of the text. On Motif it adds the gap only on the right side of the text. On GTK+ and Cocoa :gap is ignored.

Example
   This example uses the columns initarg:
(capi:contain
 (make-instance
 'capi:multi-column-list-panel
 :visible-min-width 300
 :visible-min-height :text-height
 :columns '((:title "Fruits"
 :adjust :right
 :width (character 15))
 (:title "Vegetables"
 :adjust :left
 :visible-min-width (character 30)))
 :items '(("Apple" "Artichoke")
 ("Pomegranate" "Pumkpin")))

This example uses header-args to add callbacks and independent alignment on the titles:

(defun mclp-header-callback (interface item)
 (declare (ignorable interface))
 (capi:display-message "Clicked on -a" item))

(capi:contain
 (make-instance
 'capi:multi-column-list-panel
 :visible-min-width 300
 :visible-min-height :text-height
 :columns '((:adjust :right
 :width (character 15))
 (:adjust :left
 :visible-min-width (character 30)))
 :header-args '(:items ("Fruits" "Vegetables")
 :selection-callback
 mclp-header-callback
 :alignments (:left :right))
 :items '(("Apple" "Artichoke")
 ("Pomegranate" "Pumkpin")))

This example uses column-function to implement a primitive process browser:
(defun get-process-elements (process)
  (list (mp:process-name process)
         (mp:process-whostate process)
         (mp:process-priority process)))

(capi:contain
 (make-instance 'capi:multi-column-list-panel
   :visible-min-width '(character 70)
   :visible-min-height '(character 15)
   :items (mp:list-all-processes)
   :columns '(((:title "Name" :adjust :left
                 :visible-min-width (character 30))
              (:title "State" :adjust :center
               :visible-min-width (character 20))
              (:title "Priority" :adjust :center
               :visible-min-width (character 12)))
             :column-function 'get-process-elements))

See also  collection  list-panel  list-view

multi-line-text-input-pane  

Class

Summary  A pane allowing several lines of text to be entered.

Package  capi

Superclasses  text-input-pane

Description  The multi-line-text-input-pane class behaves like a text-input-pane, except that the text entered by the user is allowed to span several lines — that is, it is allowed to contain Newline characters.

See also  text-input-pane
**non-focus-list-interface**

*Class*

| Summary | Created (and destroyed) only by `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-list-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

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 callabck for the filtering-gesture).
	non-focus-list-add-filter adds a filter is 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`.
- `gesture`
  A gesture specifier.

Values

- `result`
  A generalized boolean.

Method Signature

`non-focus-maybe-capture-gesture (non-focus-interface non-focus-list-interface) gesture`

Description

The generic function `non-focus-maybe-capture-gesture` needs to return non-nil if the gesture `gesture` was captured, which means it should not be processed any more, or `nil` if `gesture` was not captured.

`gesture` should be a gesture specifier, which is an object that can be coerced to a Gesture Spec by `sys:coerce-to-gesture-spec`.

The method on `non-focus-list-interface` does the following:

1. If the gesture is `Escape` it calls `non-focus-terminate` on the non-focus window.

2. It checks whether the gesture matches any of the gestures in the `gesture-callbacks` of the window. The gesture callbacks are either explicitly defined using the initargs `:gesture-callbacks` or `:add-gesture-callbacks`, or implicitly. By default, all the gestures that are used in in-place completion (see "In-place completion" in the *CAPI User Guide*) are defined implicitly. These include `Up`, `Down`, `PageUp`, `PageDown` (selection in the list panel), `Return` (action), `Control+Return` and `Control+Shift+Return` (control of the filter). The implicitly defined gestures are affected by `gesture-callbacks`, `filtering-gesture` and `filtering-toggle`.

If a match is found, it is invoked as described for `gesture-callbacks` in `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.
non-focus-update | Generic Function

Summary
Updates the non-focus-interface.

Signature
non-focus-update non-focus-interface

Method Signature
non-focus-update (non-focus-interface non-focus-list-interface)

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

See also
prompt-with-list-non-focus
non-focus-terminate

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

<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>
<tr>
<td>menu</td>
<td>A menu.</td>
</tr>
<tr>
<td>item-identifier</td>
<td>A string or symbol.</td>
</tr>
</tbody>
</table>

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 `I OleObject::DoVerb`).

Notes

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`

---

**ole-control-close-object**

*Function*

**Summary**

Closes the object in an `ole-control-pane`.

**Signature**

`ole-control-close-object pane`
Arguments  

 pane  

 An ole-control-pane.

Description  

 The function `ole-control-close-object` closes the object that is currently in the ole-control-pane pane.

Notes  

 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-component 

Class  

Summary  

 An implementation of the interfaces in the OLE Control protocol.

Package  

capi  

Superclasses  

com:standard-i-unknown  

Initargs  

 pane-function  

 A function that is called when OLE embeds the Control in a container.

 create-callback  

 A function called just after the pane is created.

 destroy-callback  

 A function called just before the pane is destroyed.

Readers  

 ole-control-component-pane
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.

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.

**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

Sublcasses 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.

Notes
ole-control-doc is implemented only in LispWorks for Windows. Load the functionality by (require "embed").
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**

<table>
<thead>
<tr>
<th><strong>Summary</strong></th>
<th>Implements the frame of components in an <code>ole-control-pane</code>.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Package</strong></td>
<td><code>capi</code></td>
</tr>
<tr>
<td><strong>Superclasses</strong></td>
<td><code>ole-control-doc</code></td>
</tr>
</tbody>
</table>

**Description**

The pane class `ole-control-frame` implements the frame of components, that is it supports the `I OleInPlaceFrame` 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 `I OleInPlaceSite::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 `I OleInPlaceFrame` 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`.

**Notes**

`ole-control-frame` is implemented only in LispWorks for Windows. Load the functionality by `(require "embed")`. Even though it is a subclass of `pinboard-layout`, normally you should not use the `pinboard-layout` functionality when using `ole-control-frame`.

**Example**

See the example in `examples/com/ole/simple-container/doc-viewer-pair.lisp`

**See also**

`ole-control-insert-object`

`ole-control-pane`

---

### `ole-control-i-dispatch`  

**Function**

<table>
<thead>
<tr>
<th>Summary</th>
<th>Returns the <code>com:i-dispatch</code> of the component of an <code>ole-control-pane</code>.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Signature</strong></td>
<td><code>ole-control-i-dispatch pane =&gt; result</code></td>
</tr>
<tr>
<td><strong>Arguments</strong></td>
<td><code>pane</code> An <code>ole-control-pane</code>.</td>
</tr>
<tr>
<td><strong>Values</strong></td>
<td><code>result</code> A <code>com:i-dispatch</code> or nil.</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>The function <code>ole-control-i-dispatch</code> returns the <code>com:i-dispatch</code> (that is, the <code>IDispatch</code> interface) of the component, or nil if there isn't any. The <code>com:i-dispatch</code> is the one that would be returned by <code>com:query-interface</code> on the <code>I-Ole-object</code>.</td>
</tr>
<tr>
<td><strong>Notes</strong></td>
<td>Calling <code>ole-control-i-dispatch</code> does not affect the reference count of the interface.</td>
</tr>
</tbody>
</table>
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.

**Notes**

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.

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.

Notes  
Calling ole-control-ole-object does not affect the reference count of the interface.
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

{(capi:contain
 (list
  (make-instance 'capi:ole-control-pane
    :component-name "OWC.Spreadsheet.9")))}

See examples/com/ole/simple-container/sink.lisp for a full example.

See also

attach-sink
component-name
detach-sink
interface-menu-groups
ole-control-add-verbs
ole-control-close-object
ole-control-i-dispatch
ole-control-insert-object
ole-control-ole-object
ole-control-pane-frame
ole-control-user-component
report-active-component-failure

**Function**

### ole-control-pane-frame

**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`

**Class**

### ole-control-pane-simple-sink

**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 (require "embed").

See also

attach-simple-sink
ole-control-pane

Function

ole-control-user-component

Summary

Gets and sets the user-component of an ole-control-pane.

Signature

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-control-pane pane, so after you have called (setf ole-control-user-component) you should not try to use it again or release it. Setting user-component also sets the pane’s component-name to nil.

Notes
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 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.

:image-lists
A plist of keywords and image-list objects.

:separator-item
An item that acts as a separator between other items, or nil.

:enabled-positions
A list of fixnums, or the keyword :all.

>window-styles
A list of keywords.

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.

visible-items-count is implemented only on Microsoft Windows. 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 \texttt{nil}.

If *image-function* is non-nil, it should be a function of one argument which is called with each item. The return value depends on *image-lists*. If *image-lists* contains an \texttt{image-list} for the \texttt{:normal} key, then the result of *image-function* should be 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 option-pane’s *image-list*. This is generally only useful if the image list is created explicitly. See \texttt{image-list} for more details.

Otherwise if there is no *image-list* then it should return one of:
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.

If `image-lists` is specified, it should be a plist containing the keyword `:normal` as a key. The corresponding value should be an `image-list` object. No other keys are supported at the present time. The `image-list` associated with the `:normal` key is used with the `image-function` (see above) to specify an image to display in each tab.

`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`.

On Microsoft Windows Vista and Windows 7, if `window-styles` contains the keyword `:simple-text-only`, then the `option-pane` is displayed using the UI theme and the `enabled-positions, separator-item, image-function and visible-items-count` initargs are not supported. Otherwise it is displayed without the UI theme and those options work as documented. This is a limitation in Microsoft Windows.

Notes

1. `:image-function` and `:image-lists` are currently only implemented for Microsoft Windows, GTK+ 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 (Windows only):

```
(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

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 called to redisplay the pane.

- `:drawing-mode`
  A keyword controlling quality of drawing, especially anti-aliasing of text.

- `: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.

- `: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 \texttt{nil}. The default is \texttt{nil}.

:resize-callback
A function called when the pane is resized, or \texttt{nil}. The default is \texttt{nil}.

:create-callback
A function called just after the pane is created.

:destroy-callback
A function called just before the pane is destroyed.

:use-native-input-method
Controls whether to use native input method to interpret keyboard input. Currently this has an effect only on GTK+.

:composition-callback
This is called for various events related to composition, which here means composing input characters into other characters by an input method.

Accessors
\begin{itemize}
  \item output-pane-display-callback
  \item output-pane-focus-callback
  \item output-pane-resize-callback
  \item output-pane-scroll-callback
  \item output-pane-create-callback
  \item output-pane-destroy-callback
  \item output-pane-composition-callback
\end{itemize}

Readers
\begin{itemize}
  \item output-pane-graphics-options
\end{itemize}
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**.

*draw-mode* should be either :compatible which causes drawing to be the same as in LispWorks 6.0, or :quality which causes all the drawing to be transformed properly, and allows control over anti-aliasing on Microsoft Windows and GTK+. The default value of *draw-mode* is :quality.

For more information about *draw-mode*, see "Drawing mode and anti-aliasing" in the CAPI User Guide.

*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.

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

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

In a character mapping, gesture can be simply the keyword :character, which matches any character input. For specific mappings, gesture can be a list containing a single character object char, or a list

(char)

Note: where input would match both a Gesture Spec mapping and a character mapping, the Gesture Spec mapping takes precedence.

In a button mapping, gesture should be list
(button action [modifiers]*)

where button is one of :button-1, :button-2 or :button-3 denoting the mouse buttons. action is one of :press, :release, :second-press, :third-press, :nth-press and :motion, and each modifier is one of the keywords :shift, :control, :meta and :hyper. The :meta modifier will be the Alt key on most keyboards. On Cocoa, the :hyper modifier is interpreted as the Command key for button and motion gestures. On Windows, the :hyper modifier is currently never generated, so gestures mappings using it will never be invoked. :third-press and :nth-press are supported only on Cocoa and Motif.

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

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

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

In a motion mapping, gesture can either be defined in terms of dragging a button (in which case it is defined as a button gesture with action :motion), or it can be defined for motions whilst no button is down by just specifying the keyword :motion with no additional arguments.

In a command mapping, gesture should be a command which is defined using define-command, and provides an alias for a gesture. The following commands are predefined:
(:button-3 :press) on Motif.
(:button-1 :press :control) on Mac OS X.

:control-post-menu
(:button-3 :press :control) on Microsoft Windows, Motif and Mac OS X.

:keyboard-post-menu
(:gesture-spec :f10 :shift) on Microsoft Windows, Motif and Mac OS X.

Note that it is recommended you follow the style guidelines and conventions of the platform you are developing for when mapping gestures to results.

When user input matches gesture, callback is called with standard arguments and any extra-callback-args as extra arguments. The standard arguments are the output-pane, the x cursor position, the y cursor position, and in the case of Gesture Spec, character or key mappings, the input object that matched.

Button mappings with action :press are matched on the first button click, and they pass the standard arguments to their callback. Button mappings with action :second-press and :third-press are matched on the second and third button click made in quick succession, and again they pass the standard arguments to their callback. Button mappings with action :nth-press are matched on the 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.

input-model can be set before the pane is displayed, but changes after that are ignored.

Also note that some built-in subclasses of output-pane specify their own input-model, so care should be taken when setting it. Generally an initial value supplied using the :input-model initarg will be prepended to any input-model specified by the built-in subclass (so your input gestures will override matching supplied gestures). However this is not true of editor-pane, where the :input-model initarg replaces the specified default input-model.

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).

use-native-input-method should be nil, t or :default. If use-native-input-method is not supplied, or is :default, the default is used, which is controlled by set-default-use-native-input-method. The default setting is always to use native input methods.

composition-callback is a function with signature

composition-callback pane what

where pane is the output pane and what can be one of:

:start The composition operation is starting.
:end The composition ends.
A list A plist describing the "preedit" string, which is a string containing the partial input that should be displayed while the composition is ongoing. These calls with a plist occur only when the underlying system does not display the partial input itself. Currently on Microsoft Windows the system always displays the preedit string itself, so these calls occur only on GTK+ and Cocoa.

During composition there will be repeated calls with a list, in general each time that the preedit string changes. Each call is a complete description of what needs to be displayed. The data from previous calls should be ignored.

The keys that can appear in the plist are currently:

:string-face-lists

The value is a list where each element is itself a list, where the first element is a string and the second aplist describing a face (a face plist). The strings are the strings that need to be displayed, and the face plist describing the face that the underlying GUI thinks that each string needs to be displayed. The face plist may contain any of the following keywords: :foreground, :background, :font, :bold-p, :italic-p, :underline-p. The argument string-face-lists may be nil, which means display nothing.

:cursor

The argument is an integer describing where the "cursor" should be displayed. The index is into the string that is concatenation of the strings in string-face-lists.
:selected-range

If present, the value specifies the selected range as a cons of start and length in characters. The start is an index into the string that is a concatenation of the strings in the string-face-list.

:selection-needs-face

A boolean specifying whether the selected-range should have a different face to the unselected range.

The editor uses the :start call to position the composition window at the cursor by using set-composition-placement and the calls with a list to display the partial composition string.

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.

3. In GTK+ versions before 2.12 the :start and :end calls are not reliable.

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 ~a 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)))

This example illustrates some effects of drawing-mode:
examples/capi/graphics/catherine-wheel.lisp

There are further examples in the directory
examples/capi/output-panes/.

See also
define-command
pinboard-object
scroll
set-default-use-native-input-method
set-composition-placement

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 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 page-setup-dialog. On Cocoa, passing contin-
uation 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.

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

Description This function calculates the offset required by the adjust keyword so that the pane pane is placed correctly within the available space in its parent layout. It is called by all of the layouts that inherit from x-y-adjustable-layout to interpret the values of x-adjust and y-adjust.

Typically the value of adjust will be a keyword or a list of the form (keyword n) where n is an integer. These values of adjust are interpreted as by pane-adjusted-position.

However, new methods can accept alternative values for adjust where required and can also add extra keywords. For example, grid-layout allows adjust to be a list of adjust values, and then passes the offset into this list as an additional keyword.

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

Example (setq button-panel (make-instance 'capi:button-panel :items '(1 2 3)))
See also layout
x-y-adjustable-layout

pane-adjusted-position

Generic Function

Summary Calculates how to place a pane correctly within a layout, given a minimum and maximum position.

Package capi

Signature pane-adjusted-position pane adjust min-position max-position &key &allow-other-keys

Description The pane-adjusted-position generic function calculates the position required by the adjust argument so that the pane pane is placed correctly within the available space in its parent layout, given a minimum and maximum position. It is a complementary function to pane-adjusted-offset, and the default method actually calls pane-adjusted-offset with the gap between the two positions, and then adds on the minimum position to get the new position.

The default method accepts the following values for adjust.

:top Place pane at the top of the region.
:bottom Place pane at the bottom of the region.
:left Place pane at the left of the region.
:right Place pane at the right of the region.
:center Place pane in the center of the region.
 (:top n) Place the top of pane n pixels below the top of the region.

 (:bottom n) Place the bottom of pane n pixels above the bottom of the region.

 (:left n) Place the left of pane n pixels after the left of the region.

 (:right n) Place the right of pane n pixels before the right of the region.

 (:center n) Place the center of pane n pixels below the center of the region.

 However, new methods can accept alternative values for adjust where required and can also add extra keywords. For example, grid-layout allows adjust to be a list of adjust values, and then passes the offset into this list as an additional keyword. It is preferable to add new methods to pane-adjusted-offset as these changes will be seen by the default method of pane-adjusted-position.

 Example

 (setq button-panel (make-instance 'capi:button-panel :items '(1 2 3)))

 (capi:pane-adjusted-position button-panel :center 100 200)

 (capi:pane-adjusted-position button-panel :right 100 200)

 (capi:pane-adjusted-position button-panel :left 100 200)

 See also layout
 graph-pane
 x-y-adjustable-layout

 pane-close-display

 Function

 Summary Closes the X display of a pane.
Package: capi

Signature: pane-close-display pane => closedp

Arguments: pane A CAPI element.

Values: closedp A boolean.

Description: The function `pane-close-display` closes the X display connection on which `pane` is currently displayed. This destroys all the other panes on the same connection.

`closedp` is true if the connection was closed.

Notes: `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.

Notes
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**
`pane-initial-focus` `pane-with-children` `=>` `pane`

**Signature**
`(setf pane-initial-focus)` `pane` `pane-with-children` `=>` `pane`

**Arguments**
`pane-with-children`
A pane with children.

**Values**
`pane`
A child of `pane-with-children`.

**Description**
The generic function `pane-initial-focus` returns the child of `pane-with-children` that has the input focus when `pane-with-children` is first displayed.

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

See also pane-has-focus-p

pane-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 => object, string, plist
           pane-interface-copy-p pane interface
           pane-interface-cut-object pane interface
           pane-interface-cut-p pane interface
           pane-interface-deselect-all pane interface
           pane-interface-deselect-all-p pane interface
           pane-interface-paste-object pane interface
           pane-interface-paste-p pane interface
**pane-interface-select-all** pane interface

**pane-interface-select-all-p** pane interface

**pane-interface-undo** pane interface

**pane-interface-undo-p** pane interface

**Description**

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.

**Notes**

1. These generic functions should not display a dialog or do anything that may cause the system to hang. In general this means interacting with anything outside the Lisp image, including files, databases and so on.

2. The three return values of `pane-interface-copy-object` are passed to `set-clipboard`.

**See also**

- `active-pane-copy`
- `item-pane-interface-copy-object`
- `set-clipboard`

---

**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 descendents `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**.

```lisp

(defun my-callback (pane) (capi:display-message "Callback on pane ~S." pane))

(defmethod capi:pane-popup-menu-items ((self capi:editor-pane) (interface edgraph)) (list* (self capi:editor-pane) (interface edgraph))
  (make-instance 'capi:menu-item :title "Item for My Editor Menu." :selection-callback 'my-callback) (call-next-method)))

(defmethod capi:pane-popup-menu-items ((self capi:graph-pane) (interface edgraph))
  (list (make-instance 'capi:menu-item :title "Item for My Graph Menu." :selection-callback 'my-callback) (capi:make-menu-for-pane self (call-next-method) :title "Default Graph Menu")))]

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

There is a further example in:

examples/capi/elements/pane-popup-menu-items.lisp
See also  

**make-pane-popup-menu**

---

**pane-screen-internal-geometry**  

**Function**

**Summary**

Returns the internal geometry of the monitor in which a pane's interface is displayed.

**Package**

capi

**Signature**

`pane-screen-internal-geometry pane => x, y, width, height`

**Arguments**

- `pane`  
  A CAPI pane.

**Values**

- `x`  
  An integer.

- `y`  
  An integer.

- `width`  
  A positive integer.

- `height`  
  A positive integer.

**Description**

The function `pane-screen-internal-geometry` returns the internal geometry of the "monitor" in which the interface that contains `pane` is displayed. A "monitor" is typically a physical monitor, but can be anything that the underlying GUI system considers a monitor.

`pane` must be inside an interface that is already displayed. `pane-screen-internal-geometry` returns the internal geometry of the monitor on which this interface is displayed. If the interface spreads across multiple monitors, it returns the geometry for the monitor on which the largest area of the interface is displayed.

The internal geometry of a monitor is a rectangle which excludes "system areas" like taskbars and global menu bars and so on. Examples of these include the Windows taskbar,
the Mac OS X menu bar, and the Mac OS X dock. See `screen-internal-geometry` for information about displaying CAPI windows in system areas.

$x, y, width$ and $height$ specify a screen rectangle, in which the $x$ and $y$ coordinates are offsets from the top-left of the primary monitor.

**Notes**
On GTK+ the internal geometry is of the workspace in which the interface is displayed. When there are multiple monitors these values may be incorrect. You can check the number of monitors by `screen-monitor-geometries`.

**See also**
`screen-internal-geometry`
`screen-internal-geometries`
`virtual-screen-geometry`

---

**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`.

**See also**
`editor-pane`
pane-supports-menus-with-images  Function

Summary  Tests whether a pane supports menus with images.

Signature  

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  


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 pass-
words, such that when the password is entered it is not visi-
tible 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 `#\*`.

```lisp
Example
(setq password-pane (capi:contain
  (make-instance 'capi:password-pane
    :callback
    #'(lambda (password interface)
      (capi:display-message
        "Password: ~A"
        password)))))

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

(setq password-pane
  (capi:contain
   (make-instance 'capi:password-pane
     :max-characters 5
     :text "abc"
     :overwrite-character #\$)))

(capi:password-pane-overwrite-character password-pane2)

See also  editor-pane
text-input-pane

---

### play-sound

**Function**

**Summary** Plays a loaded sound on Microsoft Windows and Cocoa.

**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`.

Notes

1. `:wait t` is only implemented on Microsoft Windows.
2. `play-sound` is not implemented on GTK+ and Motif.

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.

Description

When a `pinboard-layout` lays out its children, it positions them at the `x` and `y` specified as hints (using `:x` and `:y`), and sizes them to their minimum size (which can be specified using `:visible-min-width` and `:visible-max-width`).
By default, the `pinboard-layout` is made sufficiently large to accomodate all of its children, as specified by `fit-size-to-children` in the superclass `static-layout`. Note that results in the pinboard resizing itself automatically when objects are added, moved or removed. If you need the sizing capabilities, then use the class `simple-pinboard-layout` which surrounds a single child, and adopts the size constraints of that child.

The pinboard layout handles the display of pinboard objects itself by calculating which objects are visible in the region that needs redrawing, and then by calling the generic function `draw-pinboard-object` on these objects in the order that they are specified in the layout description. This means that if two pinboard objects overlap, the later one in the layout description will be on top of the other one. In other words, the description defines the Z-order for objects of type `pinboard-object`. For information about controlling this order, see `layout` and `manipulate-pinboard`.

**Note:** objects of type `simple-pane` are drawn directly by the windowing system and cannot be clipped relative to `pinboard-objects`, which are drawn by CAPI. Therefore `simple-panes` always appear on top in a pinboard, and their position in the description does not affect the Z-order.

Highlighting of the layout’s children by `highlight-pinboard-object` is controlled by the value of `highlight-style`, as follows:

- **:invert** Swaps the foreground and background colors.
- **:standard** Uses system colors.
- **:default** Calls `draw-pinboard-object-highlighted`.

The default value of `highlight-style` is `:default`. 
Notes

1. The output-pane initarg :drawing-mode controls quality of drawing in a pinboard-layout, including anti-aliasing of any text displayed on Microsoft Windows and GTK+.

2. 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:lispworks-file
 "examples/capi/graphics/Setup.bmp")
 :x 20 :y 20)))
 :best-width 540 :best-height 415)

(capi:contain
 (make-instance 'capi:pinboard-layout
 :description (list
 (make-instance 'capi:item-pinboard-object
 :text "Hello"
 :x 40 :y 10)
 (make-instance 'capi:line-pinboard-object
 :x 10 :y 30
 :visible-min-width 100))
 :best-width 200 :best-height 200)
```

There are further examples in the directories examples/capi/applications/ and examples/capi/graphics/.
This example illustrates use of `draw-with-buffer t`:

```
examples/capi/graphics/compositing-mode.lisp
```

See also

- `manipulate-pinboard`
- `output-pane`
- `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**

- `ellipse`
- `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.
- `:automatic-resize` A plist.

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.

:external-max-width
The maximum width of the pinboard object in the pinboard.

:external-max-height
The maximum height of the pinboard object in the pinboard.

:visible-min-width
The minimum visible width of the pinboard object.

:visible-min-height
The minimum visible height of the pinboard object.

:visible-max-width
The maximum visible width of the pinboard object.

:visible-max-height
The maximum height of the pinboard object.

:internal-min-width
The minimum width of the display region.

:internal-min-height
The minimum height of the display region.
:internal-max-width

   The maximum width of the display region.

:internal-max-height

   The maximum height of the display region.

Accessors

pinboard-object-pinboard
pinboard-object-activep
pinboard-object-graphics-args

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 graphics-state).

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:

ellipse       Draws an ellipse.
rectangle     Draws a rectangle.
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. If you want non-standard highlighting, you can implement methods for your subclass of pinboard-object.

You can test whether a whether a point or region coincides with a pinboard object by the generic functions over-pinboard-object-p and pinboard-object-overlap-p. The default methods assume a rectangle based on the geometry, which must always be the enclosing rectangle of the whole pinboard object. Therefore you only need to implement methods if your subclass of pinboard-object has a non-rectangular shape.
automatic-resize makes the pinboard object resize automatically. This has an effect only if it is placed inside a static-layout (including subclasses like pinboard-layout). The effect is that when the static-layout is resized then the pinboard object also changes its geometry.

The value of automatic-resize defines how the pinboard object’s geometry changes. It must be a plist of keywords and values which match the keywords of the function set-object-automatic-resize and are interpreted in the same way.

Notes You can also 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
over-pinboard-object-p
redraw-pinboard-object
redraw-pinboard-layout
pinboard-object-overlap-p
set-object-automatic-resize
unhighlight-pinboard-object

pinboard-object-at-position

Summary The generic function pinboard-object-at-position returns the uppermost pinboard object containing a specified point.

Package capi
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

```lisp
(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 parameter in a `pinboard-object`.

**Package**
capi

**Signature**

\[
\text{pinboard-object-graphics-arg \ self \ keyword} => \text{value}
\]

\[
\text{(setf pinboard-object-graphics-arg) value \ self \ keyword} => \text{value}
\]
### pinboard-object-graphics-arg

**Arguments**

- **self**: A pinboard-object.
- **keyword**: A keyword denoting a graphics state parameter.

**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 state parameter `keyword` in `self`.

See `graphics-state` for details of the drawing parameters.

**See also**

- `graphics-state`
- `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  \hspace{1cm} \textit{Generic Function}

\textbf{Summary} \hspace{1cm} Gets and sets the location of an object inside its parent \textit{pinboard-layout}.

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

\textbf{Signature} \hspace{1cm} \texttt{pinboard-pane-position self => x, y}

\texttt{setf (pinboard-pane-position self) (values x y) => x, y}

\textbf{Arguments} \hspace{1cm} \texttt{self} \hspace{1cm} A \texttt{pinboard-object} or \texttt{simple-pane}.

\textbf{Values} \hspace{1cm} \texttt{x, y} \hspace{1cm} The horizontal and vertical coordinates in the \texttt{pinboard-layout} parent of \texttt{self}.

\textbf{Description} \hspace{1cm} The generic function \texttt{pinboard-pane-position} returns as multiple values \texttt{x, y} the coordinates of \texttt{self} inside its parent \texttt{pinboard-layout}.

There is also a \texttt{setf} expansion which sets the location of \texttt{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)
      (dottimes (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)))))

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-size self) (values width height) =>
width, height

Description

The generic function pinboard-pane-size returns as multiple values width, height the dimensions of self.

There is also a setf expansion which 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-confirm

Summary

The **popup-confirm** 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-confirm 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-function no-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-
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-
firmer or display-dialog. In recursive calls, only the handler of the innermost call to popup-confirmer or display-dialog is established.

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

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

All other arguments will be passed to the call to make-instance for the interface that will be displayed using display-dialog. Thus geometry information, colors, and so on can be passed in here as well. By default, the dialog will pick up the foreground, background and font of pane.

Notes

1. On Windows and Motif, the effect of callback-error-handler can be achieved by using cl:handler-bind around the call to display-dialog or popup-confirmer (the handler will also handle errors during raising the dialog, but these are not expected to happen). On Cocoa, using such an error handler does not necessarily work, because the callback may happen in another process. callback-error-handler ensures that the callback is in the scope of the handler on all platforms. From the same reason the handler should not rely on the dynamic environment (including catchers and restarts), and needs to use current-popup to find its "context" and use abort-callback, exit-dialog or abort-dialog for non-local exit.

2. If the callback itself calls popup-confirmer or display-dialog, the callback-error-handler handler will stay until the callback returns. Unless the recursive call handles the error, the handler of the outer call may be called to handle it, and needs to be written to deal with this possibility correctly. If the handler inside a recursive call needs to
access the popup that was used in the same call that the handler was used, it should close over it, because current-popup returns the innermost one.

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

Example

Here are two simple examples which implement the basic functionality of two CAPI prompters: the first implements a simple prompt-for-string, while the second implements prompt-for-confirmation.

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

(capi:popup-confirmert 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-confirmert
 (make-instance 'capi:text-input-pane
   :callback 'capi:exit-confirmert
   :change-callback :redisplay-interface)
 "Enter an integer"
 :value-function 'pane-integer
 :ok-check 'integerp)
An example illustrating the use of :button-container:

(let* ((bt (make-instance 'capi:simple-layout
                         :title "Button Container"
                         :title-position :left))
       (tip1 (make-instance 'capi:text-input-pane
                         :title "Top"))
       (tip2 (make-instance 'capi:text-input-pane
                         :title "Bottom"))
       (layout (make-instance 'capi:column-layout
                         :description
                         (list tip1
                               bt
                               tip2))))

(capi:popup-confirmer layout nil
                         :title
                         "Dialog using button-container"
                         :button-container bt))

An example with all the defined buttons in use:
(defun all-buttons-dialog (&optional (num 20))
  (let ((pane
         (make-instance 'capi:list-panel
           :items
           (loop for ii from 1
to num
            collect
            (format nil "-\(\(r\) ii))
            :visible-min-width
            '(character 20))))
    (capi:popup-acceptor
     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**

*Class*

Summary: A button with a popup menu.

Package: capi

Superclasses: item

Initargs:
- :menu A menu or nil.
  
- :menu-function A function designator or nil.

Accessors: popup-menu-button-menu, popup-menu-button-menu-function

Description: 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

**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.

**print-capi-button**

*Generic Function*

Summary Generates the text for a button.

Package capi

Signature

```
print-capi-button button => text
```

Arguments

- `button` A button.

Values

- `text` A string.

Description The generic function `print-capi-button` is used to generate the text for a button.
You can add methods for your own button classes.

See also button

**print-collection-item**

*Generic Function*

Summary Prints an item as a string.
Package: capi

Signature: print-collection-item item collection

Arguments:
- item: An item or an Lisp object.
- collection: A collection or any Lisp object.

Description:
The generic function print-collection-item prints item as a string. It is used when item is known to be an item in collection.

An item in a collection prints using the first of these which returns non-nil: the item’s text, the item’s print-function, the collection’s print-function or the item’s data. An item not known to be in the collection is printed simply using print-object.

The method on (t collection) uses the collection’s print-function.

Example:
```lisp
(setq collection (make-instance 'capi:collection
 :items '(1 2 3 4 5)
 :print-function #'(lambda (x)
                     (format nil "<-A:>
                             x)))))

(capi:print-collection-item 2 collection)
```

In this example we provide our own print-collection-item method:
(defclass my-tree-view (capi:tree-view) ())

(defmethod capi:print-collection-item ((item capi:item) (tree my-tree-view))
  (string-capitalize (svref (capi:item-data item) 0)))

(capi:contain
  (make-instance 'my-tree-view
    :roots
    (list (make-instance 'capi:item
      :data
      (vector "foo")))))

See also get-collection-item
collection

print-dialog

Function

Summary Displays a print dialog and returns a printer object.

Package capi

Signature print-dialog &key screen owner first-page last-page print-
  selection-p print-pages-p print-copies-p continuation => printer

Values printer A printer, or nil.

Description The function print-dialog displays a print dialog and
returns a printer object. The printer object returned will print
multiple copies if requested by the user.

If print-pages-p is t, the user can select a range of pages to
print. This should always be the case unless the application
only produces single page output. If print-pages is t, first-page
and last-page can be used to initialize the page range. For
example, they could be set to be the first and last pages of the
document.
The `print-copies-p` argument indicates whether the application handles production of multiple copies for drivers that do not support this function. Currently this should be `nil` if the application uses Page Sequential printing and `t` if the application uses Page on Demand printing.

If `print-selection-p` is `t`, the user is given the option of printing the current selection. Only specify this if the application has a notion of selection and selecting printing functionality is provided.

The dialog is displayed on the current screen unless `screen` specifies otherwise.

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

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

Note that the printer object itself is opaque but programmatic setting of some printer options is available via the function `set-printer-options`.

See also `print-file`  
`print-text`  
`set-printer-options`

### print-editor-buffer

**Function**

**Summary**

Prints the contents of an editor buffer to the printer.
print-editor-buffer

**Description**

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

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

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

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

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

---

print-file

**Function**

**Summary**

Prints the contents of a specified file.

**Package**

`capi`

**Signature**

`print-file file &key printer interactive font`

**Description**

The `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  

\[
\text{print-rich-text-pane} \text{ pane} \&\text{key} \text{ jobname} \text{ printer} \text{ interactive} \\
\text{selection} => \text{result}
\]

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>jobname</td>
<td>A string, or nil.</td>
</tr>
<tr>
<td>printer</td>
<td>A printer, or nil.</td>
</tr>
<tr>
<td>interactive</td>
<td>A boolean.</td>
</tr>
<tr>
<td>selection</td>
<td>A boolean.</td>
</tr>
</tbody>
</table>

Values  

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

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.

Notes  

print-rich-text-pane is supported only on Microsoft Windows.

See also  
rich-text-pane
Function

**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`

Function

**printer-configuration-dialog**

**Summary**
Displays a dialog allowing the user to configure printers.

**Package**
capi
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 Class

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
printer-port-handle

Function

Summary
Returns the underlying handle to a printer port.

Package
capi

Signature
printer-port-handle &optional port => handle

Arguments
port A printer port.

Values
handle Platform-dependent.

Description
The function `printer-port-handle` returns a platform-dependent value which represents the underlying handle to the printer port.

On Microsoft Windows, `handle` is the HDC for the printer device.

If `port` is passed it should be the value bound to `var` in `with-print-job`. If `port` is not supplied it defaults to the current printer port (dynamically bound within `with-print-job`).

See also
`with-print-job`

printer-port-supports-p

Function

Summary
Detects if the printer port can support a certain feature.

Package
capi

Signature
printer-port-supports-p feature &optional port => supportedp, validp

Arguments
feature A keyword.
port A printer port.
Values

- **supportedp**: A boolean.
- **validp**: A boolean.

Description

The function `printer-port-supports-p` detects if the printer port can support the feature named by `feature`.

If `port` is passed it should be the value bound to `var` in `with-print-job`. If `port` is not supplied it defaults to the current printer port (dynamically bound within `with-print-job`).

- `supportedp` indicates if the feature is supported.
- `validp` indicates if the feature was recognised.

Currently the only value of `feature` that is recognised is `:postscript` and the `supportedp` value is true if the printer supports PostScript.

See also `with-print-job`

`*printer-search-path*`  

**Variable**

Summary

Specifies where to look for printer definition files.

Package

- `capi`

Initial value

- `("~/.lispworks-printers/" nil)`

Description

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

This applies only on GTK+ and Motif.

The value is a list containing directory pathname designators specifying where to look for printer definition files. The list can also include the value `nil`, which is interpreted as the `printers` directory in the LispWorks library.
To find known printers the system loads all files in these directories. If there are duplicate printer definitions, the printer in the first directory takes precedence.

The default path is useful when printing from the Common LispWorks IDE, but applications that want to allow users to use printers should set the list appropriately.

The first path in the *printer-search-path* list is regarded as the "local" path. New printers are saved in this path. When the user edits a printer that was found in another directory on *printer-search-path* and then tries to save it, the system prompts for whether to overwrite the original or save it in the "local" directory.

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

A printer file can be copied to other machines, provided the printer is installed on the other machine and the PPD file is available in the same path.

---

**process-pending-messages**

*Function*

**Summary** Processes all the pending messages in the current process.

**Package** 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

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 integers delimiting the range of values the progress bar can display.

The accessor (setf range-slug-start) is used to set an integer value for the progress indicator.

See also
range-pane
titled-object

prompt-for-color

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 CAPI User Guide.

`owner` specifies an owner window for the dialog. See the "Prompting for Input" chapter in the 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`.
Values

- `result` A boolean.
- `successp` A boolean.

Description

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

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

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

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

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

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

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

The prompt itself is created by passing an appropriate pane to **popup-confirmor**. Arguments can be passed to the *make-instance* of the pane and the call to **popup-confirmor** using *pane-args* and *popup-args* respectively. Currently, the pane used to create the file prompter is internal to the CAPI.
See also

popup-confirmerprompt-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 Source Files" "*.LISP;*.LSP"
 "Lisp Fasls" "*.OFASL"
 "Text Documents" "*.DOC;*.TXT"
 "Image Files" "*.BMP;*.DIB;*.ICO;*.CUR"
 "All Files" "**.**")
```

The "Lisp Fasls" extension may vary depending on the implementation.

On Cocoa the default value of `filters` is:

```
("Lisp Source Files" "*.lisp;*.lsp"
 "Text Documents" "*.txt;*.text"
 "All Files" "**.**")
```

`filters` is ignored on Motif.
When if-exists is :ok, an existing file can be returned. Otherwise the user is prompted about whether the file can be overwritten. The default for if-exists is :ok when operation is :open and :prompt when operation is :save.

When if-does-not-exist is :ok, a non-existent file can be chosen. When it is :prompt, the user is prompted if a non-existent file is chosen. When it is :error, the user cannot choose a non-existent file. The default for if-does-not-exist is :prompt if operation is :open and :ok if operation is :save.

operation chooses the style of dialog used, in LispWorks for Windows only. The default value is :open.

owner, if non-nil, specifies an owner window for the dialog. See the "Prompting for Input" chapter in the 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-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. 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

```lisp
(capi:prompt-for-file "Enter a filename:" filepath "")

(capi:prompt-for-file "Enter a filename:" path:pathname "/usr/bin/cal")

(capi:prompt-for-file "Enter a filename:" path:ok-check 'probe-file)
```

See also
- `popup-confirmers`
- `prompt-for-string`
- `prompt-for-directories`

**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.
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:

```lisp
("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.

**Notes**

`prompt-for-files` is currently implemented only in Lisp-Works for Windows and Cocoa.

**See also**

`prompt-for-file`
Signature  

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 CAPI User Guide for details.

For a description of Graphics Ports fonts and font descriptions, see the CAPI User Guide.

See also  

find-best-font

prompt-for-form  

Function

Summary  

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-confirmer` using `pane-args` and `popup-args` respectively, and an input history can be implemented by supplying a `history-function` or `history-symbol` in `popup-args`.

Example

Try the following examples, and each time enter `(+ 1 2)` into the input pane.

(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-confirmer`
- `text-input-pane`
**prompt-for-forms**

**Function**

**Summary**
Displays a text input pane prompting the user for a number of forms.

**Package**
capi

**Signature**

\[
\text{prompt-for-forms \ message \ &key \ package \ initial-value \ value-function \ pane-args \ popup-args \ continuation} \Rightarrow \text{result, okp}
\]

**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.

\[
\text{(capi:prompt-for-forms \ "Enter some forms:")}
\]
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 \texttt{ok-check} function. \texttt{ok-check} should take one argument, the currently entered number, and should return \texttt{t} if it is valid. If \texttt{ok-check} is \texttt{nil} (the default) then there is no further restriction.

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

The prompter is created by passing \texttt{text-input-pane} to \texttt{popup-confirm}. Arguments can be passed to the \texttt{make-instance} of the pane and the call to \texttt{popup-confirm} using \texttt{pane-args} and \texttt{popup-args} respectively.

\begin{verbatim}
Example
\begin{verbatim}
(capi:prompt-for-integer "Enter an integer:")
(capi:prompt-for-integer "Enter an integer:" :max 10)
(capi:prompt-for-integer "Enter an integer:
  :min 100 :max 200)
(capi:prompt-for-integer "Enter an integer:
  :ok-check 'evenp)
\end{verbatim}
\end{verbatim}

See also
\begin{verbatim}
  prompt-for-string
  popup-confirm
  text-input-pane
\end{verbatim}

\textbf{prompt-for-items-from-list} \hspace{1cm} \textit{Function}

\textbf{Summary} \hspace{1cm} Prompts with a choice of items.

\textbf{Package} \hspace{1cm} \texttt{capi}
1 CAPI Reference Entries

**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`. 
ok-check  A function or nil.
pane-args  Arguments to pass to the pane.
popup-args  Arguments to pass to the confirmer.
continuation  A function or nil.

Description  The function prompt-for-number pops up a text-input-pane and prompts the user for a number, which is returned in result.

The functionality corresponds exactly to that of prompt-for-integer, except that all types of numbers are allowed.

See also  prompt-for-integer

prompt-for-string  

Function

Summary  Displays a text input pane and prompts the user for a string.

Package  capi

Signature  prompt-for-string message &key pane-args popup-args ok-check value-function text initial-value print-function history-symbol history-function continuation => result, okp

Description  The function prompt-for-string prompts the user for a string and returns that string in result and a flag okp indicating that the dialog was not cancelled. The initial string can either be supplied directly as a string using the text argument, or by passing initial-value and a print-function for that value. print-function defaults to princ-to-string. The value returned can be converted into a different value by passing a value-function, which by default is the identity function. This value-function gets passed the text that was entered into the pane, and should return both the value to return and a flag
that should be non-nil if the value that was entered is not acceptable. If an \texttt{ok-check} is passed, then it should return non-nil if the value about to be returned is acceptable.

\texttt{prompt-for-string} creates an instance of \texttt{text-input-pane} or \texttt{text-input-choice} depending on the value of \texttt{history-function}. Arguments can be passed to the \texttt{make-instance} of this pane using \texttt{pane-args}. \texttt{prompt-for-string} then passes this pane to \texttt{popup-确认mer}. Arguments can be passed to the call to \texttt{popup-确认mer} using \texttt{popup-args}.

\texttt{history-symbol}, if non-nil, provides a symbol whose value is used to store an input history, when \texttt{history-function} is not supplied. The default value of \texttt{history-symbol} is \texttt{nil}.

\texttt{history-function}, if supplied, should be a function designator for a function with signature:

\begin{verbatim}
history-function &optional push-value
\end{verbatim}

\texttt{history-function} is called with no argument to obtain the history which is used as the \texttt{items} of the \texttt{text-input-choice}, and with the latest input to update the history.

The default value of \texttt{history-function} is \texttt{nil}. In this case, if \texttt{history-symbol} is non-nil then a history function is constructed which stores its history in the value of that symbol.

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-string}. On Cocoa, passing \texttt{continuation} causes the dialog to be made as a window-modal sheet and \texttt{prompt-for-string} returns immediately, leaving the dialog on the screen. The \texttt{with-dialog-results} macro provides a convenient way to create a \texttt{continuation} function.

\textbf{Example} \begin{verbatim}
(capi:prompt-for-string "Enter a string:")
\end{verbatim}
(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-confirm
        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:"
 :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`
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-confirmers. Arguments can be passed to the make-instance of the pane and the call to popup-confirmers using pane-args and popup-args respectively.

**Example**

```lisp
(capi:prompt-for-value
 "Square"
 :initial-value '(+ 1 2 3)
 :value-function
 #'(lambda (text)
      (let ((res (eval (read-from-string text)))))
      (* res res)))
```
See also  prompt-for-form

**prompt-with-list**

**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 buttons callbacks all-button none-button => 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.
- `buttons`  A list of strings or the keyword `:none`.
- `callbacks`  A list of callback specs.
- `all-button`  A string, nil or t.
- `none-button`  A string, nil or t.

**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.

class determines the type of choice used in the dialog. 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. Note that interaction :multiple-selection is not supported for lists on Mac OS X.

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.

In addition to the choice showing the items, prompt-with-list can also display a panel of push buttons (the "action buttons") which perform actions related to the choice. Note that these buttons are separated from the "dialog buttons" such as OK and Cancel. The dialog buttons are controlled separately by keywords in popup-args.
By default, `prompt-with-list` does not display action buttons. However, if `interaction` is `:multiple-selection`, the default behavior is to display two action buttons, **All** and **None**. These change the selection to all of the items or none of the items respectively.

When `buttons` is **:none**, it specifies no action buttons in any case (including no **All** and **None** buttons). Otherwise `buttons` must be a list of strings specifying additional action buttons. Each of the strings specifies a button, and the string is displayed in the button.

`callbacks` specifies the callbacks of the buttons. It should be a list of callback specifiers matching the list in `buttons`. Each callback specifier is either a callable (a function or a symbol) which takes one argument, the choice, or a list where the **car** is a callable which is called as follows:

```
(apply (car callback-spec) choice (cdr callback-spec))
```

When `all-button` and `none-button` are supplied they override the default behavior of the **All** and **None** buttons. If `all-button (none-button)` is nil, then **All (None)** is not displayed. If `all-button (none-button)` is non-nil and `buttons` is not **:none**, the **All (None)** button is displayed, and if the value is string, that string is used instead of the default string.

The prompter is created by passing an appropriate pane (in this case an instance of class `choice-class`) to `popup-confirmer`. Arguments can be passed to the `make-instance` of the pane and the call to `popup-confirmer` using `pane-args` and `popup-args` respectively. The initial selection can be specified using `choice initargs` **:selection, :selected-item** or **:selected-items** in `pane-args`.

Example

```
(capi:prompt-with-list
  '((1 2 3 4 5) "Select an item:"
```
(capi:prompt-with-list
 '(1 2 3 4 5) "Select some items:"
 :interaction :multiple-selection
 :selection '(0 2 4))

(capi:prompt-with-list
 '(1 2 3 4 5) "Select an item:"
 :interaction :multiple-selection
 :choice-class 'capi:button-panel)

(capi:prompt-with-list
 '(1 2 3 4 5) "Select an item:"
 :interaction :multiple-selection
 :choice-class 'capi:button-panel
 :pane-args
   '(:layout-class capi:column-layout))

There is a more complex example in
examples/capi/choice/prompt-with-buttons.lisp

See also
popup-confirmers
list-panel
choice

prompt-with-list-non-focus

Function

Summary

Raises a non-focus window.

Signature

prompt-with-list-non-focus items &key owner x y choice-class
vertical-scroll print-function selection selected-item visible-items
selection-callback action-callback destroy-callback list-updater
gesture-callbacks add-gesture-callbacks alternative-y alternative-x
alternative-bottom alternative-right widget-name filtering-gesture
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 . callback).

add-gesture-callbacks  A list of pairs of the form (gesture . callback).

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

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

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

gesture-callbacks and add-gesture-callbacks define gesture callbacks which the non-focus window can "capture" (when non-focus-maybe-capture-gesture is called). gesture-callbacks and add-gesture-callbacks should both be a list of pairs of the form (gesture . callback). Each gesture must be a gesture specifier, that is an object that systo: coerce-to-gesture-spec can coerce to a Gesture Spec. Each callback is either a callable (symbol or function) which takes one argument, the non-focus window, or a list of the form (function arguments).

Note that when it is a list, the window is not automatically
passed to the function `function` amongst the arguments `arguments`. The gesture callbacks are used only when `non-focus-maybe-capture-gesture` is called.

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

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

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

`widget-name` has an effect only on Motif. It defines the widget name of the interface, which can then be used to define resources specific to the non-focus window. Note that the non-focus completers in `editor-pane` and `text-input-pane` use the default `widget-name` which is "non-focus-list-prompter", so defining resources for non-focus-list-prompter will affect them.
If `items` is `nil`, `prompt-with-list-non-focus` returns `nil` without doing anything. Otherwise, it raises the non-focus window and returns the interface, which is of class `non-focus-list-interface`.

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

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

See also
- `list-panel`
- `non-focus-terminate`
- `non-focus-update`
- `non-focus-list-toggle-filter`
- `non-focus-list-add-filter`
- `non-focus-list-remove-filter`
- `non-focus-list-toggle-enable-filter`
- `non-focus-maybe-capture-gesture`

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`. 
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

```lisp
(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.

Notes

callback (from superclass button) is the general callback, triggered when the user clicks the button, either by pressing and releasing the mouse button or by a keyboard gesture.

press-callback is called only when the user presses the mouse 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

**Superclasses**
button-panel

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

**Example**

```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

```
(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 An integer specifying the lowest value of the range.
:end An integer specifying the highest value of the range.
:slug-start An integer specifying the start of the slug, corresponding to the current value of the range.
:slug-end An integer specifying 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

**Summary**
Reads data from a sound file on Microsoft Windows and Cocoa.

**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.

**Notes**
1. `read-sound-file` can be called during image building.
2. `read-sound-file` is not implemented on GTK+ and Motif.
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.

The rectangle is always drawn with shape-mode :plain (that is, without anti-aliasing).

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 redisplay-item redisplay 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.

Notes This method is called by popup-confirmers 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 &key redo-items

Arguments interface An interface.
**redo-items**  
A generalized boolean.

**Description**  
The function `redisplay-menu-bar` updates the interface’s menu bar, such that menus become enabled and disabled as appropriate.

When `redo-items` is non-nil, `redisplay-menu-bar` redoes the items in `menu` and `menu-component` that have an `items-function`, by calling the `items-function` and setting the items. The default value of `redo-items` is `t`.

**Notes**  
`redo-items` defaults to `t` in order to ensure that any accelerator associated with any item is up-to-date. When the menu bar contains menus (including sub-menus and menu-components) that have an `items-function`, `redisplay-menu-bar` may take a relatively long time (tens of milliseconds). If it is called often (for example, each time the user types a character), then it is better to call `redisplay-menu-bar` with `redo-items nil`.

**Compatibility note**  
This function has been superseded by `redisplay-interface`, which updates the menu bar, but also updates other state objects such as buttons, list panels and so on.

**See also**  
`interface`  
`redisplay-interface`

---

**redraw-pinboard-layout**  
*Function*

**Summary**  
Redraws any pinboard objects within a specified rectangle.

**Package**  
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**  

**Function**

**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/`.

**See also**
- `pinboard-object`
- `pinboard-layout`
- `redraw-pinboard-layout`

---

**reinitialize-interface**  

**Generic Function**

**Summary**  
Reinitializes an existing `interface`.

**Package**  
capi
reinitialize-interface interface &rest initargs

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")
```
remove-items

**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 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.
replace-dialog

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.

See also
display-replacable-dialog

replace-items

Summary
Replaces 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 collection from items. replace-items can only be used when the collection has the default items-get-function svref.

start should be a non-negative integer and less than the number of items in collection.

Items in collection are replaced starting at index start, and proceeding until the end of the list items, or the end of the items in collection. If items is too long, the surplus is quietly ignored. replace-items never alters the number of items in the collection.

If supplied, new-selection should be a list of items specifying the new selection in collection. To specify no selection, pass nil.

If new-selection is not supplied, then replace-items attempts to preserve the selection. If some of the selected items are replaced, then the selection on these items is removed, but if a selected item simply moves, then the selection moves with it.

See also
append-items
collection
remove-items

report-active-component-failure                  Generic Function

Summary Reports on failures to find or create a component.

Package capi
Signature report-active-component-failure pane component-name error-string function-name hresult

Arguments

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

Description

The generic function `report-active-component-failure` is used to report on failures to find or create a component. `component-name` is the name of the component it tried to find. `error-string` is the error string. `function-name` is the name of the function that actually failed. `hresult` is the hresult that came back. It may be `nil` if the error is that the guid of the named component could not be found.

When the system fails to open the component, it calls `report-active-component-failure`, with the first argument the `ole-control-pane` `pane`. The default method for `ole-control-pane` tries to call `report-active-component-failure` again on its top level interface. The default method on `interface` calls `error`.

You can add your own methods, specializing on subclasses of `ole-control-pane` or subclasses of `interface`.

Notes

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

Example

See the example in `examples/com/ole/simple-container/doc-viewer-pair.lisp`

See also `ole-control-pane`
reuse-interfaces-p

Function

Summary
Determines whether global interface re-use is enabled.

Package
capi

Signature
reuse-interfaces-p => result

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

Arguments
value A boolean.

Values
result A boolean.

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

The function (setf reuse-interfaces-p) enables or disables global interface re-use.

If global re-use is enabled, then locate-interface and find-interface may return existing interfaces. If global re-use is disabled, then locate-interface returns nil and find-interface returns a new interface.

See also
find-interface
locate-interface

rich-text-pane

Class

Summary
A text pane with extended formatting.

Package
capi

Superclasses
simple-pane
Initargs

:character-format
   A plist.

:paragraph-format
   A plist.

:change-callback
   A function called when a change is made.

:protected-callback
   A function determining whether the user may edit a protected part of the text, on Microsoft Windows.

: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.

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, if supplied, is called when the user tries to modify protected text (by setting the protected attribute, see set-rich-text-pane-character-format). It must be 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. If protected-callback is not supplied, then the user cannot modify protected text. 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.

Notes

1. 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 above.

2. change-callback and protected-callback are not yet implemented on Cocoa.

3. 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.
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 Must be t. This argument is deprecated.

Values
result A plist.

Description
The function **rich-text-pane-character-format** returns as a plist the current character attributes for **pane**.

If there is a current selection in the pane, then the attributes are those set for the selected text. If there is no selection, then it gets the "typing attributes", which are applied to characters that are typed by the user. Note that any cursor movement changes these attributes, so their values are ephemeral.

The **selection** argument is deprecated. If **selection** is **nil** an error is signalled. 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:
• \texttt{t} if all the selection is bold
• \texttt{nil} if all the selection is not bold
• \\texttt{:unknown} if the selection is only partially bold.

For the possible attributes, see \texttt{set-rich-text-pane-character-format}.

\textbf{Compatibility note} \hfill The value \texttt{nil} for the keyword argument \texttt{:selection} is not supported in LispWorks 6.1 and later. See the description above for details of the current behavior with respect to the current selection in the \texttt{rich-text-pane}.

\textbf{See also} \hfill \texttt{rich-text-pane} \hfill \texttt{set-rich-text-pane-character-format}

\textbf{rich-text-pane-operation}{
\hspace*{1cm} \textit{Function}}

\textbf{Summary} \hfill Gets and sets values and performs various operations on the pane.

\textbf{Package} \hfill capi

\textbf{Signature} \hfill \texttt{rich-text-pane-operation pane operation \&rest args} => \texttt{result, result2}

\textbf{Arguments} \hfill \begin{align*} 
\texttt{pane} & \quad \text{A \texttt{rich-text-pane}.} \\
\texttt{operation} & \quad \text{A keyword specifying the operation to perform.} \\
\texttt{args} & \quad \text{The value or values to use, when the operation is setting something.} 
\end{align*}

\textbf{Values} \hfill \begin{align*} 
\texttt{result} & \quad \text{Various, see below.} 
\end{align*}
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

---

rich-text-pane-paragraph-format

Function

Summary

Returns the paragraph format.

Package
capi

Signature

rich-text-pane-paragraph-format pane => result

Arguments

pane A rich-text-pane.
### rich-text-pane-paragraph-format

**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: :type The type of line.

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:

```
(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 arguments 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 :separator, you should use nil to specify that the divider or separator 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:

(capi:contain
  (make-instance 'capi:row-layout
    :description (list (make-instance 'capi:push-button
      :text "foo")
      nil
      (make-instance 'capi:push-button
        :text "bar")
    :ratios 'nil
    :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

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
Function

**screen-internal-geometries**

**Summary**
Returns the internal geometries of all the monitors of a screen.

**Package**
capi

**Signature**
`screen-internal-geometries screen => internal-geometries`

**Arguments**
- `screen` A CAPI screen.

**Values**
- `internal-geometries` A list of screen rectangles.

**Description**
The function `screen-internal-geometries` returns the internal geometries of all the "monitors" of screen. A "monitor" typically corresponds to a physical monitor, but can be anything that the underlying GUI system considers a monitor.

The internal geometry of a monitor is a rectangle which excludes "system areas" like taskbars and global menu bars and so on. Examples of these include the Windows taskbar, the Mac OS X menu bar, and the Mac OS X dock. See `screen-internal-geometry` for information about displaying CAPI windows in system areas.

Each internal geometry is represented as a screen rectangle. A screen rectangle is a list of four numbers: `x` and `y` being the coordinates as offsets from the top-left of the primary monitor, and `width` and `height`.

The first screen rectangle in the `internal-geometries` list corresponds to the usable area of the primary monitor.

**Notes**
On GTK+ when using a desktop with separate workspaces, the workspaces may be considered as separate "monitors". When there are multiple real monitors, the values may be
incorrect. You can use `screen-monitor-geometries` to check the number of monitors, and to check the full size of the monitors.

See also
- `pane-screen-internal-geometry`
- `virtual-screen-geometry`
- `screen-internal-geometry`
- `screen-monitor-geometries`

### screen-monitor-geometries

**Function**

**Summary**
Returns the geometries of all of a screen's monitors.

**Package**
capi

**Signature**

`screen-monitor-geometries screen => monitor-geometries`

**Arguments**

- `screen` A CAPI screen.

**Values**

- `monitor-geometries` A list of screen rectangles.

**Description**

The function `screen-monitor-geometries` returns the geometries of all the monitors of `screen`. A monitor corresponds to an entity that the host machine regards as a physical monitor. `screen-monitor-geometries` ignores software manipulations like the desktop on GTK+.

The monitor geometry is a rectangle which includes all of its display area, including "system areas" like menubar and taskbar and so on. Examples of these include the Windows taskbar, the Mac OS X menu bar and the Mac OS X dock.

Each monitor geometry screen rectangle is represented by a list of four numbers: the `x` and `y` coordinates as offsets from the top-left of the primary monitor, and the `width` and `height`.

The first screen rectangle in the `monitor-geometries` list corresponds to the primary monitor.
Notes

1. `screen-monitor-geometries` differs from `screen-internal-geometries` by returning screen rectangles which include all the monitor areas, and also by ignoring desktop manipulations.

2. You cannot display a CAPI window on the Mac OS X menu bar. You can display a CAPI window in the area occupied by the Mac OS X dock or the Windows task bar, but the window will be obscured.

See also

`pane-screen-internal-geometry`
`screen-internal-geometries`
`virtual-screen-geometry`

**screen-internal-geometry**

*Function*

**Summary**

Returns the geometry of the unobscured region of a screen or document container.

**Package**

`capi`

**Signature**

`screen-internal-geometry screen => x, y, width, height`

**Arguments**

`screen` A screen.

**Values**

`x` An integer.

`y` An integer.

`width` A positive integer.

`height` A positive integer.

**Description**

The function `screen-internal-geometry` returns the geometry (as multiple values representing a screen rectangle) of the region of the screen that can be used to display windows without obstruction. This region excludes "system
areas” like menubar and taskbar and so on. Examples of these include the Windows taskbar, the Mac OS X menu bar and the Mac OS X dock.

$x$ and $y$ are the screen rectangle’s coordinates as offsets from the top-left of the primary monitor, and $width$ and $height$ are its dimensions.

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

**Notes**

1. The internal geometry is a snapshot of the unobscured region of a screen. If a system area moves or changes size, then the screen rectangle returned by `screen-internal-geometry` changes.

2. It may be possible to display a CAPI window outside the screen’s internal geometry, for example under the Mac OS X dock, but it will be obscured.

3. The primary monitor is that represented by the first screen rectangle in the list returned by `screen-internal-geometries`.

**See also**

- `document-container`
- `pane-screen-internal-geometry`
- `screen`
- `screen-internal-geometries`

**screens**

**Function**

**Summary**

Returns the active screens for a library.

**Package**

`capi`

**Signature**

`screens &optional library => result`
Arguments

| library | A library name, a list, or :any. |

Values

| result | A list. |

Description

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

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

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

See also

default-library

display

**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
scroll-bar

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, it 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 (ignore 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  
\textit{selection self &optional format => result}

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

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

Description  
The function \textit{selection} returns the contents of the primary selection as a string, or \texttt{nil} if there is no selection.

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

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

When \textit{format} is \texttt{:image}, the image returned by \textit{selection} is associated with \textit{self}, so you can free it explicitly with \texttt{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 \texttt{interprogram-paste-function}.

See also  
\texttt{clipboard} 
\texttt{free-image} 
\texttt{image} 
\texttt{selection-empty} 
\texttt{set-selection}
selection-empty

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.

Description
The function `selection-empty` returns nil if there is a primary selection of the kind indicated by `format`, or t if there is no such selection.

`format` controls what kind of object is checked. The following values of `format` are recognized:

- `:string` The object is a string. This the default value.
- `:image` The object is of type `image`, converted from whatever format the platform supports.
- `:value` The object is the Lisp value.

See also
image
selection

set-application-interface

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.

An application can only have one application menu and one dock menu. Because the LispWorks IDE already provides these menus, calling `set-application-interface` while running the LispWorks IDE will add a submenu to the LispWorks application menu to contain the `application-menu` and `menu-bar-items` of your application, and you can test them there. Likewise, a submenu will be added to the LispWorks Dock icon menu. Other aspects of the application interface can only be tested when running it standalone.

`sset-application-interface` is only applicable when running under Cocoa.

Example  

See these files in the `examples` subdirectory of the LispWorks library:

- `capi/applications/cocoa-application.lisp`
- `capi/applications/cocoa-application-single-window.lisp`
- `delivery/macos/multiple-window-application.lisp`
- `delivery/macos/single-window-application.lisp`

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**

\[
\text{set-button-panel-enabled-items} \quad \text{button-panel} \quad \&\text{key} \quad \text{enable} \\
\text{disable} \quad \text{set} \quad \text{test} \quad \text{key}
\]

**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**

\[
\text{set-clipboard} \quad \text{self} \quad \text{value} \quad \&\text{optional} \quad \text{string} \quad \text{plist} \quad \Rightarrow \quad \text{result}
\]

**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 as described for clipboard. You can export more than one format simultaneously.

**Values**  

| **result** | A string, or nil. |

**Description**  

The function set-clipboard sets the contents of the system clipboard to be the text of 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 xclipboard or the Emacs function x-get-clipboard. The most likely explanation for apparent inconsistencies after 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:

```lisp
(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

Function

set-composition-placement

Summary Specifies the placement of the composition window relative to the pane. Composition here mean composing input characters into other characters by an input method.

Signature set-composition-placement pane x y &key width height force

Description The function set-composition-placement tells the system where to place the composition window in pixel coordinates relative to the pane pane.

On systems where the composition text is displayed by the application (rather than by the system, when the composition callback is called with a plist), the placement coordinates are used to place the composition menu when it is raised.

x and y are the top left coordinates. If both width and height are supplied, they specify the dimensions of the composition window. If force is supplied with a true value, the coordinates are forced, overriding adjustments that the system may otherwise do.

x, y and, when supplied, width and height must all be positive integers.

Notes set-composition-placement does not raise the composition window. It merely tells the system where to place the composition window when it does appear.

See also output-pane
**set-confirm-quit-flag**  
*Function*

**Summary**
Controls the behavior of *confirm-quit*

**Package**
capi

**Signature**
`set-confirm-quit-flag flag`

**Arguments**
`flag` One of t, nil or :check-editor-files

**Description**
The function `set-confirm-quit-flag` sets a flag which controls the behavior of *confirm-quit*.

See `confirm-quit` for the effect.

**Note:** on initialization, the LispWorks IDE sets the flag to the stored value of the option Tools > Preferences... > Environment > General > Confirm Before Exiting.

See also `confirm-quit`

---

**set-default-editor-pane-blink-rate**  
*Function*

**Summary**
Sets the default cursor blinking rate for editor panes.

**Package**
capi

**Signature**
`set-default-editor-pane-blink-rate 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-default-use-native-input-method  

Function
Summary  
Controls the default of using native input method on GTK+.

Signature  
set-default-use-native-input-method &key output-pane
editor-pane => t

Arguments  
output-pane  A boolean.
editor-pane  A boolean.

Values  
set-default-use-native-input-method returns t.

Description  
The function set-default-use-native-input-method controls whether the native input method is used by default. Currently it has an effect only on GTK+.

The values of the keyword arguments are booleans. editor-pane changes the default for editor-pane and subclasses. output-pane controls the default for output-pane and subclasses, except editor-pane and its subclasses.

If a keyword argument is not supplied, the corresponding default is not set.

See also  
output-pane
editor-pane

set-display-pane-selection  

Generic Function
Summary  
Sets the selection in a display-pane.

Package  
capi
Signature

\texttt{set-display-pane-selection \textit{pane} \textit{start} \textit{end}}

Arguments

\begin{description}
\item[\textit{pane}] A \texttt{display-pane}.
\item[\textit{start}, \textit{end}] Bounding indexes for a subsequence of the text of \textit{pane}.
\end{description}

Description

The generic function \texttt{set-display-pane-selection} sets the selection in \textit{pane} to be the text bounded by the indexes \textit{start} (inclusive) and \textit{end} (exclusive).

See also

\begin{description}
\item[display-pane-selection]
\item[display-pane]
\end{description}

\textbf{set-drop-object-supported-formats} \hspace{1cm} \textit{Function}

Summary

Sets the list of formats for a drop object

Package

capi

Signature

\texttt{set-drop-object-supported-formats \textit{drop-object} \textit{formats}}

Arguments

\begin{description}
\item[\textit{drop-object}] A \texttt{drop-object}, as passed to the \texttt{drop-callback}
\item[\textit{formats}] A list of format keywords
\end{description}

Description

The function \texttt{set-drop-object-supported-formats} sets the list of formats that the drop object \textit{drop-object} wants to receive.

The \texttt{:string} format can be used to receive a string from another application and the \texttt{:filename-list} format can be used to receive a list of filenames from another application such as the Macintosh Finder or the Windows Explorer.
GTK+ supports dragging of list of URIs. LispWorks uses a list of URIs to pass/receive the data with the format :filename-list, and also adds the format :uris. The behavior is as follows:

- For dragging with format :filename-list (that is, call drag-pane-object with a plist containing :filename-list, or including :filename-list in the value that drag-callback returns) the argument must be a list of pathname designators. LispWorks canonicalizes the pathnames and converts them to file URIs.

- For dragging with format :uris, each value in the list must either a string containing a colon, or a pathname designator. A string containing a colon is passed unchanged. Other it is assumed to be a pathname designator, and is converted to a file URI.

- For dropping with format :filename-list (that is, calling drop-object-get-object with :filename-list), LispWorks converts each file URI to the corresponding filename string (without checking whether it is a proper file name), and discards all other URIs.

- For dropping with format :uris, LispWorks returns all the URIs as strings.

There is an example of :filename-list and :uris in examples/capi/elements/gtk-filename-list-and-uris.lisp

On Cocoa and GTK+ the :image format can be used to receive images. The value passed needs to be an image obect.

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.
Notes  
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  
examples/capi/choice/drag-and-drop.lisp  
examples/capi/choice/list-panel-drag-images.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 CAPI User Guide for information about colors.

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

If colors is t or nil, parenthesis coloring is switched on or off, without changing the list of colors.
When parenthesis coloring is off, parentheses are drawn like other characters.

See also editor-pane

**set-geometric-hint**

*Function*

**Summary**
The `set-geometric-hint` function sets the hint associated with a key.

**Package**
capi

**Signature**
```
set-geometric-hint element key value
&optional override
```

**Description**
Set the hint associated with `key` to `value`. If `override` is `nil`, the value is not changed when there is already a hint for this key. The default is `t`.

See also

- set-hint-table
- element

**set-hint-table**

*Function*

**Summary**
Modifies the hint table for an element.

**Package**
capi

**Signature**
```
set-hint-table element plist
```

**Description**
The function `set-hint-table` modifies the hint table for the element `element` to include `plist`. All existing hints are retained for keys not in the `plist`.

---

526
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.
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**

Sets the break gestures on GTK+ and Motif.

**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-list-panel-keyboard-search-reset-time**

**Function**

**Summary**
Sets the default length of time before resetting the "last match" in keyboard searching in a list-panel.

**Signature**

`set-list-panel-keyboard-search-reset-time time`

**Arguments**

`time` A positive real number.

**Description**

The function `set-list-panel-keyboard-search-reset-time` sets the default length of time before resetting the "last match" in keyboard searching in a list-panel. The argument `time` specifies this time in seconds.

When the user types a character into a list-panel, if there is a "last match" the system searches for a string made of the "last match" followed by the character, otherwise it searches for a string made of the character only. The system sets the "last match" when it matches, and remembers the "last
match" for one second by default. `set-list-panel-keyboard-search-reset-time` can be used to change the time for which the "last match" is kept.

Notes
When `keyboard-search-callback` returns a third value non-nil, the value that `set-list-panel-keyboard-search-reset-time` sets is ignored.

See also
`list-panel`
`list-panel-search-with-function`

**set-object-automatic-resize**

**Function**

**Summary**
Controls automatic resizing and repositioning of objects in a static layout.

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

A real number, default value 0.5.

pinboard

A static-layout, if supplied. This argument is deprecated, and can always be omitted.

Description

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

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

object should be either a pinboard-object or a simple-pane which is (or will be) displayed in a static-layout. This object will be added to the description of the layout by one of its :description initarg, (setf capi:layout-description) or manipulate-pinboard.

pinboard is the 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
   
   \( \frac{1}{\frac{1}{\text{calculated-height}} \times \text{calculated-width}} \times \text{aspect-ratio} \)

3. Compute correction as
   
   \( \text{expt \ aspect-ratio-ratio \ aspect-ratio-y-weight} \)

4. Compute the new width as calculated-width multiplied by correction, and the new height as the new width multiplied by aspect-ratio.

The result is that if aspect-ratio-y-weight is 0, correction is 1 and height-ratio is effectively ignored, while if aspect-ratio-y-weight is 1, correction cancels the effect of width-ratio. With the default value of 0.5, the resulting position is in the (geometric) middle, and object takes a fixed fraction of the area of the pinboard.
After resizing (if needed), object is also positioned horizontally if \texttt{x-align} is non-nil, and vertically if \texttt{y-align} is non-nil.

The new \texttt{x} coordinate of \texttt{object} is calculated as follows:

- If \texttt{x-ratio} is set, the new \texttt{x} coordinate is the sum of \texttt{x-ratio} multiplied by the width of \texttt{pinboard} plus \texttt{x-offset}, otherwise it is simply \texttt{x-offset}.

- The actual value of the \texttt{x} coordinate for \texttt{object} is adjusted according to the value of \texttt{x-align} such that the left, center or right of \texttt{object} align with the new coordinate.

The new \texttt{y} coordinate of \texttt{object} is calculated similarly, using \texttt{y-ratio} and \texttt{y-offset}, with an adjustment such that the top, center or bottom of \texttt{object} aligns with the new coordinate according to \texttt{y-align}.

If all of \texttt{width-ratio}, \texttt{height-ratio}, \texttt{x-align} and \texttt{y-align} are \texttt{nil}, automatic resizing/re-positioning of \texttt{object} is removed.

\texttt{set-object-automatic-resize} can be called before \texttt{object} is actually displayed, and its effect persists over calls adding and removing \texttt{object} to/from \texttt{static-layouts}. The effect of \texttt{set-object-automatic-resize} also persists if \texttt{object} is removed and added again, either to the same layout or another layout.

Repeated calls to \texttt{set-object-automatic-resize} set only the values that are passed to \texttt{set-object-automatic-resize}. Keys that are not passed are left with their previous value. A call that removes the automatic resizing (because \texttt{width-ratio}, \texttt{height-ratio}, \texttt{x-align} and \texttt{y-align} are all \texttt{nil}) erases all the values.

\texttt{set-object-automatic-resize} returns \texttt{t} if the object is set up for automatic resizing, or \texttt{nil} if the object is set up for no automatic resizing.

Notes

1. The initarg \texttt{:automatic-resize} can be used to set up automatic resizing in the call to \texttt{make-instance}.  

533
2. The name `set-object-automatic-resize` is slightly inaccurate, because this function can alter an object’s position without actually changing its size.

**Compatibility note**

In LispWorks 6.0 the effect of `set-object-automatic-resize` does not persist if the object is removed and then added, to any layout.

In LispWorks 6.0 each call to `set-object-automatic-resize` sets all the values.

**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 layout 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)
```

There is a further example in

```
(example-file "capi/layouts/automatic-resize.lisp")
```

**See also**

`manipulate-pinboard`

`static-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**
`pane`  
A `rich-text-pane`.

`selection`  
Must be t. This argument is deprecated.

`attributes-plist`  
A plist or :default.

**Values**
`result`  
A plist.

**Description**
The function `set-rich-text-pane-character-format` sets current character attributes for text in `pane`. 
If there is a current selection in the pane, then the attributes are set for the selected text. If there is no selection, then it sets the "typing attributes", which are applied to characters that are typed by the user. Note that any cursor movement changes these attributes, so the setting is ephemeral.

The selection argument is deprecated. If selection is nil an error is signalled. The default value of selection is t.

If attributes-plist is the symbol :default then the default character format of the pane (that is, the value of the rich-text-pane initarg :character-format) 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. See the description of protected-callback in rich-text-pane.
: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 \texttt{(logior pitch family)} where pitch and family have the value of a Windows pitch and a Windows font family respectively.

Compatibility note

The value \texttt{nil} for the keyword argument \texttt{:selection} is not supported in LispWorks 6.1 and later. See the description above for details of the current behavior with respect to the current selection in the \texttt{rich-text-pane}.

Example

\textbf{Note:} This example uses some features which are supported only on Microsoft Windows:

\begin{verbatim}
(defun ok-to-edit-p (pane start end s)
  (declare (ignore pane))
  (capi:prompt-for-confirmation
   (format nil "Editing-:[ ~; selection ~]from -a to -a" 
           s start end)))

(setq rtp
  (capi:contain
   (make-instance 'capi:rich-text-pane :protected-callback 'ok-to-edit-p :character-format '(:size 14  :color :red) :visible-min-height 300 :visible-min-width 400 :paragraph-format '(:start-indent 20 :offset -15) :text-limit 160 :text (format nil "First paragraph.~%Second paragraph, a little longer.~%Another paragraph, which should be long long enough that it spans more than one line. ~%" ))))
\end{verbatim}

Enter some characters in the rich text window and select a range.

Set the selection to blue:
(capi:set-rich-text-pane-character-format rtp
 :attributes-plist '(:color :blue))

Make it protected:

(capi:set-rich-text-pane-character-format rtp :attributes-plist '(:protected t))

Now try to delete a character, and also to delete the selection. In both cases the \texttt{ok-to-edit-p} callback is called.

\textbf{See also}  \texttt{rich-text-pane}  \\
\texttt{rich-text-pane-character-format}

---

\textbf{set-rich-text-pane-paragraph-format} \hspace{1cm} \textit{Function}

\textbf{Summary}  Sets the paragraph format.

\textbf{Package}  \texttt{capi}

\textbf{Signature}  \texttt{set-rich-text-pane-paragraph-format pane attributes-plist => result}

\textbf{Arguments}  \texttt{pane} A \texttt{rich-text-pane}.

\texttt{attributes-plist} A plist, or \texttt{:default}.

\textbf{Values}  \texttt{result} A plist.

\textbf{Description}  The function \texttt{set-rich-text-pane-paragraph-format} sets paragraph attributes for the current paragraphs in \texttt{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:

- `: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 \textit{indent} + \textit{offset} is negative, then these lines are not indented.

\textit{tab-stops} should be a list of numbers specifying the locations of tabs. No more than 32 tabs are allowed.

\textbf{Example}

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

\textbf{See also}
\begin{itemize}
  \item rich-text-pane
  \item rich-text-pane-paragraph-format
\end{itemize}

\textbf{set-selection}  \hspace{1cm}  \textit{Function}

\textbf{Summary}  \hspace{1cm}  Sets the primary selection.

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

\textbf{Signature}  \hspace{1cm}  \textit{set-selection} \textit{self} \textit{value} &optional \textit{string} \textit{plist} => \textit{result}

\textbf{Arguments}  \hspace{1cm}  \begin{itemize}
  \item \textit{self} \hspace{1cm}  A displayed CAPI pane or interface.
  \item \textit{value} \hspace{1cm}  A Lisp object (not necessarily a string) to make available within the local Lisp image.
\end{itemize}
**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:

```lisp
(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

<table>
<thead>
<tr>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summary</td>
</tr>
<tr>
<td>Package</td>
</tr>
<tr>
<td>Signature</td>
</tr>
<tr>
<td>Description</td>
</tr>
</tbody>
</table>
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:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nil</td>
<td>Print directly to the device.</td>
</tr>
<tr>
<td>t</td>
<td>Print to a file chosen by the user at printing time.</td>
</tr>
<tr>
<td>A pathname</td>
<td>Print to the file given by pathname.</td>
</tr>
</tbody>
</table>

Values of first-page are:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>:all</td>
<td>Print all pages.</td>
</tr>
<tr>
<td>A integer</td>
<td>Print from this page to the page given by last-page.</td>
</tr>
</tbody>
</table>

Values of orientation are:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>:landscape</td>
<td>Print in landscape mode.</td>
</tr>
<tr>
<td>:portrait</td>
<td>Print in portrait mode.</td>
</tr>
</tbody>
</table>

Values of copies:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A integer</td>
<td>The number of copies to print.</td>
</tr>
</tbody>
</table>

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

```lisp
;; 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  
Generic 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.

$x$ and $y$ are measured from the top-left of the screen rectangle representing the area of the primary monitor (the primary screen rectangle).

**Notes**
On Cocoa `set-top-level-interface-geometry` behaves as if an interface toolbar is not present, even if *interface* does contain an interface toolbar.

**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
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:
Class

shell-pane

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`:

```lisp
(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:

```lisp
(defun send-keys-to-pane (pane string newline-p)
  (capi:apply-in-pane-process pane
   'send-keys-to-pane-aux
   pane string newline-p))
```

```lisp
(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`:

```lisp
(send-keys-to-pane sp "dir" t)
```

### show-interface

**Function**

| 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**  
*Function*

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

Subclasses  
switchable-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 x-gap and y-gap are 200 and 100 respectively.

*simple-network-pane* is a subclass of choice, so for details of its selection handling, see choice.

**Example**  
See the file examples/capi/graphics/network.lisp.

### simple-pane

**Class**

**Summary**  
The class simple-pane is the superclass for any elements that actually appear as a native window, and is itself an empty window.

**Package**  
capi

**Superclasses**  
element
Subclasses

display-pane
interface
title-pane
button-panel
list-panel
option-pane
output-pane
progress-bar
slider
text-input-pane
tree-view
toolbar
layout
button

Initargs

:enabled A boolean controlling whether the pane is enabled.

:background The background color of the pane.

:foreground The foreground color of the pane.

:font The default font for the pane.

:horizontal-scroll
t, :without-bar, or nil. If true the pane can scroll horizontally.

:vertical-scroll
t, :without-bar, or nil. If true the pane can scroll vertically.

:visible-border A boolean or a keyword controlling whether the pane has a border, for some pane classes.

:internal-border A non-negative integer, or nil. Controls the width of the internal border.

:cursor A keyword naming a built-in cursor, or a cursor object, or nil.
:pane-menu  Specifies a menu to be raised by the :post-menu gesture.

:drop-callback  Specifies a drop callback for output-pane, interface, list-panel or tree-view. Note that this is now supported for list-panel and tree-view on Cocoa and GTK+.

:drag-callback  Specifies a drag callback for list-panel or tree-view.

:automatic-resize  A plist.

: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.

Accessors
simple-pane-enabled  
simple-pane-background  
simple-pane-foreground  
simple-pane-font  
simple-pane-cursor  
simple-pane-scroll-callback  
simple-pane-drop-callback  
simple-pane-drag-callback

Readers
simple-pane-horizontal-scroll  
simple-pane-vertical-scroll  
simple-pane-visible-border

Description  enabled determines whether the pane is enabled. The default value is t. Note that changing the enabled state of a visible pane changes its appearance.
background and foreground are colors specified using the Graphics Ports color system. Additionally on Cocoa, the special value :transparent is supported, which makes the pane’s background match that of its parent.

font should be a font, a font-description, or nil. If it is not a font, it is converted to a font when the pane is created. nil is converted to the default font, and a font-description is converted as if by calling find-best-font.

The value for visible-border can be any of the following, with the stated meanings where applicable:

nil Has no border.

 t Has a border.

 :default Use the default for the window type.

 :outline Add an outline border.

There are various platform/pane class combinations which do not respond to all values of visible-border. For instance, on Windows XP with the default theme, text-input-choice and option-pane always have a visible border regardless of the value of visible-border, while other classes including display-pane, text-input-pane, list-panel, editor-pane and graph-pane have three distinct border styles, with visible-border :default meaning the same as visible-border t.

If internal-border is non-nil, it should be a non-negative integer specifying the width of an empty region around the edge of the pane.

Any simple pane can be made scrollable by specifying t to :horizontal-scroll or :vertical-scroll. By default these values are nil, but some subclasses of simple-pane default them to t where appropriate (for instance editor-panes always default to having a vertical scroll-bar).
For a pane which is scrollable but does not display a scroll bar, pass the value :without-bar for :horizontal-scroll or :vertical-scroll. See the example in output-panes/scrolling-without-bar.lisp.

The height and width of a scrollable simple pane can be specified by the initargs :scroll-height and :scroll-width, which have the same meaning as :internal-min-height and :internal-min-width. See the CAPI User Guide for more information about height and width initargs.

cursor specifies a cursor for the pane. nil means use the default cursor, and this is the default value. cursor can also be a cursor object as returned by load-cursor. The other allowed values are keywords naming built-in cursors which are supported on each platform as shown in the table below.

<table>
<thead>
<tr>
<th>cursor</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>
<tr>
<td>:top-side</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>:bottom-side</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>:wait</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>:crosshair</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>:gc-notification</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>:top-left-corner</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>:top-right-corner</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Table 1.2
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 CAPI User Guide.

drop-callback can be specified for a pane that is an instance of output-pane, interface, list-panel, tree-view or a sub-class of one of these. When the user drags an object over a window, the CAPI first tries to call the drop-callback of any pane under the mouse and otherwise calls the drop-callback of the top-level interface. The default value of drop-callback is nil, which means that there is no support for dropping into the pane.

For editor-pane, drop-callback can be :default, which provides support for dropping a string into the pane and inserting the string into the pane's editor buffer.

If drop-callback is any other non-nil value, it should be either a list (for simple cases) or function designator (to use all options). When it is a function designator, it needs to have this signature:

<table>
<thead>
<tr>
<th>cursor</th>
<th>Cocoa</th>
<th>Windows</th>
<th>Motif</th>
</tr>
</thead>
<tbody>
<tr>
<td>:bottom-left-corner</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>:bottom-right-corner</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>:hand</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>:fleur</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>:move</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>:closed-hand</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>:open-hand</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>:disappearing-item</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

Table 1.2
The function `drop-callback` is called by the CAPI at various times such as when the pane is displayed and when the user attempts to drop data into the pane. `pane` is the pane itself, `drop-object` is an object used to communicate information about the current dropping operation (see below) and `stage` is a keyword. `drop-callback` should handle these values of `stage`:

- **:formats** This might occur when the pane is being displayed or might occur each time the user drags or drops an object over the pane. It should call `set-drop-object-supported-formats` with the `drop-object` and a list of formats that the pane wants to receive. Each format is a keyword. The list of the formats must be the same each time it is called.

- **:enter** This occurs when the user drags an object over the pane which is an `output-pane` or `interface` (not `tree-view` or `list-panel`). It can query the `drop-object` using `drop-object-provides-format` and `drop-object-allows-drop-effect-p` to discover what the user is dragging. It can also use `drop-object-pane-x` and `drop-object-pane-y` to query the mouse position relative to the pane. It should call `(setf drop-object-drop-effect)` with an effect if it wants to allow the object to be dropped. If this is not called, then the object cannot be dropped into the pane.

- **:drag** This occurs while the user is dragging an object over the pane. It can query the `drop-object` using `drop-object-provides-format` and `drop-object-allows-drop-effect-p` to discover what the user is dragging. For `output-pane`, it can use `drop-
object-pane-x and drop-object-pane-y to query the mouse position relative to the pane. For list-panel and tree-view, it can use drop-object-collection-index or drop-object-collection-item to query where the user is attempting to drop the object and can call their setf functions to adjust this position. It should call (setf drop-object-drop-effect) with an effect if it wants to allow the object to be dropped. If this is not called, then the object cannot be dropped into the pane. For output-pane and interface, it might also want to update the pane to indicate where the object will be dropped.

:drop This occurs when the user drops an object over the pane. It can query the drop-object as for the :drag stage, but can also obtain the object itself using drop-object-get-object for one of the formats in the list returned by drop-object-provides-format. Once the object is received, it should call (setf drop-object-drop-effect) with the effect that has been used by the callback. It should also update the pane to incorporate the object in whatever way the application requires.

When drop-callback is a list, it specifies a simple response. The list should be of the form:

(effects formats drop-stage-callback &optional checker)

Both effects and formats can be either a list of effects or formats, or an atom which is interpreted as a list of one element. effects and formats specify which effects and formats are allowed.
For the stages except :formats, the first effect of the given effects that the drop-object allows is set (by calling (setf drop-object-drop-effect)), except when checker is supplied. In the latter case, before setting an effect it loops through the formats and calls the checker with three arguments:

```
funccall checker pane effect format
```

If checker returns non-nil it sets the effect. If checker returns nil for the formats, it goes to the next effect.

In the :drop stage, after setting the effect, it gets the object with first format that is provided by the drop-object, and then calls the drop-stage-callback with four arguments:

```
funccall drop-stage-callback pane object x-or-index y-or-placement
```

If the pane is a tree-view or list-panel, the last two arguments are the item index (for get-collection-item) and placement (:above, :item, :below), which are the results of drop-object-collection-index. Otherwise, the last two arguments are the x and y (results of drop-object-pane-x and drop-object-pane-y). It is the responsibility of the drop-stage-callback to perform whatever dropping should mean.

drag-callback can be specified for a pane that is an instance of list-panel or tree-view. The default value of drag-callback is nil, which means that there is no support for dragging from the pane. Otherwise, it should be a function designator with this signature:

```
drag-callback pane info => result
```

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-
The `drag-callback` should normally return a plist `result` 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.

The plist `result` returned by `drag-callback` can contain the key `:image-function` with a function `image-function` as value.

This function is used to generate the image that is used in the dragging itself, exactly as the `image-function` in `drag-pane-object` is used. On Cocoa, `tree-view` and `list-panel` ignore this key in `result`.

`drag-callback` can also be used in top-level interfaces. In this case the second argument `info` is a flag describing the gesture that caused the call. Currently the only value is `:drag-image`, which means it was invoked by dragging the `drag-image` (see `interface`).

`drag-callback` is allowed to return the result `:default` rather than a plist. `:default` tells the system to do default dragging if there is any. At the time of writing the only place where there is default dragging is on Cocoa for an interface with an `:interface-pathname`. `drag-callback` is allowed to return the result `nil`, meaning do not do dragging.

On `output-pane` you add dragging by adding an entry to the `input-model` and which initiates the dragging by calling `drag-pane-object`.

`automatic-resize` makes the pane resize automatically. This has an effect only if it is placed inside a `static-layout` (including subclasses like `pinboard-layout`). The effect is that when the `static-layout` is resized then the pane also changes its geometry.
The value of `automatic-resize` defines how the pane’s geometry changes. It must be a plist of keywords and values which match the keywords of the function `set-object-automatic-resize` and are interpreted in the same way.

`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`.

3. On Cocoa in Mac OS X 10.2, the only supported `cursor` is `:i-beam`.

4. If `:image` is supplied in the `plist` returned by `drag-callback`, the dragging mechanism automatically frees the `image` object as if by `free-image` when it no longer needs it.

5. You can also control automatic resizing of a `simple-pane` using `set-object-automatic-resize`. 
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/choice/drag-and-drop.lisp

See also

contain
set-object-automatic-resize

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**

```lisp
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 CAPI User Guide for a description of the visible size of a pane.

See also `simple-pane-visible-size`

### `simple-pane-visible-size`

**Summary** Gets the visible size of a pane.

**Package** `capi`
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 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 *CAPI User Guide* for a description of the visible size of a pane.

See also  
- `simple-pane-visible-height`
- `simple-pane-visible-size`
- `with-geometry`

### simple-pinboard-layout

**Class**

**Summary**

A `simple-pinboard-layout` is a `pinboard-layout` that can contain just one pinboard object or pane as its child, and it adopts the size constraints of that child.

**Package**

`capi`

**Superclasses**

- `pinboard-layout`
- `simple-layout`

**Subclasses**

- `graph-pane`

**Initargs**

`:child`  
The child of the pinboard layout.

**Description**

The class `simple-pinboard-layout` is normally used to place pinboard objects in a layout by placing the layout inside a `simple-pinboard-layout`, thus displaying the pinboard objects. It inherits all of its layout behavior from `simple-layout`. 
Example

```
(setq column
  (make-instance
   'capi:column-layout
   :description
   (list
    (make-instance
     'capi:image-pinboard-object
     :image
     (sys:lispworks-file
      "examples/capi/graphics/Setup.bmp"))
    (make-instance
     'capi:item-pinboard-object
     :text "LispWorks"))
   :x-adjust :center))

(capi:contain (make-instance
  'capi:simple-pinboard-layout
  :child column))
```

See also  pinboard-object

---

**simple-print-port**

*Function*

**Summary**

Prints the contents of an output pane to a printer.

**Package**

capi

**Signature**

```
simple-print-port port &key jobname scale dpi printer drawing-mode 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`. 

---

565
drawing-mode should be either :compatible which causes drawing to be the same as in LispWorks 6.0, or :quality which causes all the drawing to be transformed properly, and allows control over anti-aliasing on Microsoft Windows and GTK+. The default value of drawing-mode is :quality.

For more information about drawing-mode, see "Drawing mode and anti-aliasing" in the CAPI User Guide.

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
:print-function
   A function of two arguments, or a format string.

:show-value-p A generalized boolean.

:start-point A keyword.

:tick-frequency
   An integer, a ratio or the keyword :default.

Accessors slider-print-function
Readers

- slider-show-value-p
- slider-start-point
- slider-tick-frequency

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, on Microsoft Windows and GTK+. The default value is `t`. `show-value-p` is ignored on Cocoa.

`start-point` specifies which end of the slider is the start point in the range. The values allowed depend on the orientation of the slider. For horizontal sliders, `start-point` can take these values:

- `:left` The start point is on the left.
- `:right` The start point is on the right.
- `:default` The start point is at the default side (the left).

For vertical sliders, `start-point` can take these values:

- `:top` The start point is at the top.
- `:bottom` The start point is at the bottom.
- `:default` The start point is at the default position, which is the top on Microsoft Windows and Motif, and the bottom on Cocoa.

`tick-frequency` specifies the spacing of tick marks drawn on the slider. If `tick-frequency` is `:default`, then the slider may or may not draw tick marks according the OS conventions. If `tick-frequency` is `0`, then no tick marks are drawn. If `tick-frequency` is a ratio `1/N` for integer `N>1`, then tick marks are drawn to divide the slider range into `N` sections. Otherwise `tick-frequency` should be an integer greater than `1` which specifies the spacing of tick marks in units between `start` and `end`. The default value of `tick-frequency` is `:default`.

`print-function`, when supplied, should be a function with signature
print-function pane value => result

where pane is the slider pane, value is its current value, and result is a string or nil. When the slider pane displays the current value, it calls print-function and displays the value as result, unless that is nil, in which case the value is printed normally.

As a special case, print-function can also be a string, which is used as the format string in a call to format with one additional argument, the value, that is

(format nil print-function value)

and the result of this call to format is displayed.

Notes

1. :print-function is not implemented on Motif.
2. :print-function has no effect on Cocoa because the slider pane never displays the value
3. Use of the print-function is determined when the slider pane is displayed. Setting the print-function in a slider that did not have a print-function when it was first displayed does not work until the slider is destroyed and displayed again. Therefore, if you want to display a slider without a print-function but set it later, initially you should supply a print-function that always returns nil, for example:

(make-instance 'capi:slider
   :start 10 :end 34
   :print-function 'false)

4. print-function is useful for displaying fractional values or values that grow logarithmically (or any other non-linear function), because the actual values in a slider are always integers that increase linearly as the slider moves.
5. On Windows the slider's value is displayed (when show-value-p is true) in a tooltip that is visible only while the user moves the marker with a mouse.
Compatibility note

In LispWorks 6.0 and earlier versions, ticks are drawn as if `tick-frequency` is `:default`.

Example

Given the default `start` and `end` of 0 and 100, this gives ticks at 0, 25, 50, 75 and 100:

```lisp
(make-instance 'slider :tick-frequency 25)
```

whilst this gives ticks at 0, 20, 40, 60, 80 and 100:

```lisp
(make-instance 'slider :tick-frequency 1/5)
```

This example illustrates the use of `print-function` to display fractional and non-linear values ranges:

capi/elements/slider-print-function.lisp

---

**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 sub-class.
- `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-sorted-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-sorted-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.

Description

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.

See also

- `sort-object-items-by`
- `sorted-object`
- `sorted-object-sorted-by`

**sorted-object-sorted-by**

*Function*

Summary

Returns the current sorting type and reverse flag of a `sorted-object`.

Package

`capi`

Signature

`sorted-object-sorted-by pane => sort-type, reversed`
1 CAPI Reference Entries

Arguments  
pane  
An instance of sorted-object or a subclass.

Values  
sort-type  
A sort type.
reversed  
A boolean.

Description  
The function sorted-object-sorted-by returns the current sorting type sort-type and reverse flag reversed of pane.  

sort-type is the type of one of the sort descriptions of pane.  
reversed is true if the pane is sorted in reverse order and false if it is sorted in normal order.

See also  
sorted-object  
sorted-object-sort-by

start-gc-monitor  
Function

Summary  
Starts a Lisp Monitor window.

Package  
capi

Signature  
start-gc-monitor screen => result

Arguments  
screen  
A screen.

Values  
result  
A boolean.

Description  
The function start-gc-monitor starts a Lisp Monitor window (otherwise known as the GC or Garbage Collector monitor) on the screen screen.  

result is t if it started a Lisp monitor, and nil if a Lisp monitor was already running on screen.

Note that this works only on Motif. There is no Lisp Monitor window on other platforms.
On Motif, `start-gc-monitor` is called automatically when the LispWorks IDE starts, but you can call `stop-gc-monitor` and `start-gc-monitor` any time.

See also `stop-gc-monitor`

---

**static-layout**

*Class*

**Summary**

A layout that allows its children to be positioned anywhere within itself.

**Package**

capi

**Superclasses**

layout

**Subclasses**

pinboard-layout

**Initargs**

`:fit-size-to-children`

A generalized boolean.

**Description**

The class `static-layout` is a layout that allows its children to be positioned anywhere within itself.

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

If `fit-size-to-children` is true, the `static-layout` is made sufficiently large to accommodate all of its children, and grows if necessary when a child is added. This is the default behavior. Otherwise the static layout has a minimum size of one pixel by one pixel which is not affected by the size of its children. If you need the sizing capabilities, then use the class `simple-layout` which surrounds a single child, and adopts the size constraints of that child.
Example

Here is an example of a static layout placing simple panes at arbitrary positions inside itself.

```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`.

574
See also  start-gc-monitor

**stop-sound**

*Function*

**Summary**
Stops a sound from playing.

**Signature**
stop-sound sound

**Arguments**
sound A sound object returned by load-sound.

**Description**
The function stop-sound stops the sound sound from playing.

See also  play-sound

**switchable-layout**

*Class*

**Summary**
A subclass of simple-layout that displays only one of its children at a time, and provides functionality for switching the displayed child to one of the other children.

**Package**
capi

**Superclasses**
simple-layout

**Initargs**
visible-child
The currently visible pane from the children.

combine-child-constraints
A generalized boolean.

**Readers**
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

```lisp
(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  

**switchable-layout-switchable-children**

---

### Generic Function

**switchable-layout-switchable-children**

**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**

---

### Class

**tab-layout**

**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.
  :image-function Returns an image for an item, on Microsoft Windows.
  :image-lists A plist of keywords and image-list objects, on Microsoft Windows.

Accessors
  tab-layout-visible-child-function

Readers
  tab-layout-combine-child-constraints
  tab-layout-image-function
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.

If **image-lists** is specified, it should be a plist containing the keyword **:normal** as a key. The corresponding value should be an **image-list** object. No other keys are supported at the present time. The **image-list** associated with the **:normal** key is used with the **image-function** to specify an image to display in each tab.
The `image-function` is called on an item to return an image associated with the item. It can return one of the following:

A pathname or string
   This specifies the filename of a file suitable for loading with `load-image`. Currently this must be a bitmap file.

A symbol
   The symbol must have been previously registered by means of a call to `register-image-translation`.

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

An image locator object
   This allowing a single bitmap to be created which contains several button images side by side. See `make-image-locator` for more information. On Microsoft Windows, it also allows access to bitmaps stored as resources in a DLL.

An integer
   This is a zero-based index into the tab-layout's `image-list`. This is generally only useful if the image list is created explicitly. See `image-list` for more details.

**Notes**

`image-lists` and `image-function` are implemented only on Microsoft Windows.

**Example**

The following example shows the use of the switchable mode of `tab-layout`. Each tab is linked to an output pane by pairing them in the `items` list.
(defun switchable-tab-layout ()
  (let* ((red-pane (make-instance 'capi:output-pane :background :red))
         (blue-pane (make-instance 'capi:output-pane :background :blue))
         (tl (make-instance 'capi:tab-layout :items (list (list "Red" red-pane) (list "Blue" blue-pane)) :print-function 'car :visible-child-function 'second))
    (capi:contain tl)))

(switchable-tab-layout)

Here is an example of the callback mode of tab-layout, which uses the selection of a tab to change the nodes of a graph pane through the selection-callback.

(defun non-switchable-tab-layout (tabs)
  (let* ((gp (make-instance 'capi:graph-pane))
         (tl (make-instance 'capi:tab-layout :description (list gp) :items tabs :visible-child-function nil :key-function nil :print-function (lambda (x) (format nil "~R" x)) :callback-type :data :selection-callback #'(lambda (data) (setf (capi:graph-pane-roots gp) (list data))))))
    (capi:contain tl)))

(non-switchable-tab-layout '(1 2 4 5 6))
See also callbacks
simple-layout
switchable-layout
tab-layout-panes
tab-layout-visible-child

**tab-layout-panes**

*Function*

**Summary**
Returns the panes in a 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

Superclasses  
choice
text-input-pane

Initargs  
:visible-items-count
An integer specifying the maximum length of the drop-down list, or the symbol :default.

:popup-callback
A function called just before the drop-down list appears, or nil.

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.
Compatibility note

In LispWorks 6.0 and earlier versions the `text-input-pane` initarg value `enabled :read-only` is not supported for `text-input-choice` on Microsoft Windows. This restriction is removed for LispWorks 6.1 and later versions.

Examples

See examples/capi/elements/text-input-choice.lisp.

See also

`choice`
`text-input-pane`

---

text-input-pane  

Class

Summary

The class `text-input-pane` is a pane for entering a single line of text.

Package

capi

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 change-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.
:search-field
Along with the next four initargs, this is implemented only on Cocoa. It specifies that the pane has "recent-items", which also means using NSSearchField.
:recent-items
See :search-field above.
:recent-items-name
See :search-field above.
:maximum-recent-items
See :search-field above.
:recent-items-mode
See :search-field above.

Accessors
text-input-pane-text
text-input-pane-max-characters
text-input-pane-completion-function
text-input-pane-callback
text-input-pane-confirm-change-function
text-input-pane-change-callback
text-input-pane-navigation-callback
text-input-pane-editing-callback
text-input-pane-enabled
text-input-pane-buttons-enabled

Readers
text-input-pane-caret-position

Description
The class text-input-pane provides a great deal of flexibility in its handling of the text being entered. It starts with the initial text and caret-position specified by the arguments text
and caret-position respectively. It limits the number of characters entered with the max-characters argument (which defaults to nil, meaning there is no maximum).

If enabled is nil, the pane is disabled. If enabled is :read-only, then the pane shows the text and allows it to be selected without it being editable. In this case the visual appearance varies between window systems, but often the text can be copied and the caret position altered. If enabled is any other true value, then the pane is fully enabled. The default value of enabled is t.

A completion-function can be specified which will get called when the completion gesture is made by the user (by pressing the Tab key) or when text-input-pane-complete-text is called. The function should have signature:

completion-function pane string => completions, start, end

where pane is the text-input-pane itself and string is the string to complete. When completion is invoked completion-function is called with pane and a string containing the text of pane to the left of the cursor.

The completion-function is called with the pane and the text to complete and should return either nil, the completed text as a string or a list completions of candidate completions. In the latter case, the CAPI will prompt the user for the completion they wish, and this will become the new text. In addition, the completion-function can return two more values, start and end, which specify a range in the text that is to be replaced if the completion is successful.

in-place-completion-function tells the pane to do in-place completion and specifies the function to use. The function should have signature:

in-place-completion-function pane string => completions, start, end
where pane is the text-input-pane itself and string is the string to complete. When in-place completion is invoked inplace-completion-function is called with pane and a string containing the text of pane to the left of the cursor.

completions needs to be a list of strings that are possible completions, a single string that is a unique completion, or the symbol :destroy. :destroy means that the in-place completion needs to stop and close the in-place window. In addition, the completion function can return two more values, start and end, which specify a range in the text that is to be replaced if the completion is successful. The function is called repeatedly whenever there is a change to the text that should be completed.

Note: If inplace-completion-function needs some dynamic information, it can put it in a property of the pane (using 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 CAPI User Guide for the user interaction.

file-completion, if non-nil, tells the pane to do file completion using an in-place window. See "In-place completion" in the CAPI User Guide for the interaction.

If file-completion is a pathname designator, its location is used as the root path for the completion.

Note: :file-completion overrides :in-place-completion-function.

Note: The behavior of in-place completion is somewhat different from other completion.
Note: The initargs :directories-only and :ignore-file-suffices can be used to change the behavior of the completion.

The default value of file-completion and in-place-completion-function is nil.

in-place-filter takes effect only when either in-place-completion-function or file-completion is non-nil. If in-place-filter is t then the in-place window can have a filter. Note that the filter needs to requested by a user gesture. Control+Return is the default in-place filter gesture. The default value of in-place-filter is t.

directories-only takes effect only if file-completion is used. If directories-only is t then in-place completion shows only directories. The default value of directories-only is nil.

ignore-file-suffices takes effect only if file-completion is used. It tells in-place completion to ignore files whose file namestring (the result of cl:file-namestring) ends with any of the strings in the list ignore-file-suffices. If ignore-file-suffices is :default, then completion uses the default value, which is the value of editor:*ignorable-file-suffices* (see config/a-dot-lispworks.lisp).

callback, if non-nil, is called when the user presses Return, unless navigation-callback is non-nil, in which case navigation-callback is called instead. If the pane has "recent-items" (implemented only on Cocoa) then the timing of calls to callback is modified: see the discussion of recent-items below for the details.

When the text or caret-position is changed, the callback change-callback is called with the text, the pane itself, the interface and the caret-position. The arguments that are passed to the change-callback can be altered by specifying the change-callback-type (see the callbacks class for details of possible values).
Note: the change-callback is potentially called more than once for each user gesture.

With the Motif implementation it is possible to check changes that the user makes to the text-input-pane by providing a confirm-change-function which gets passed the new text, the pane itself, its interface and the new caret position, and which should return non-nil if it is OK to make the change. If nil is returned, then the pane will be unaltered and a beep will be signalled to indicate that the new values were invalid.

navigation-callback, if non-nil, is a function that will be called when certain navigation gestures are used in the text-input-pane. The function is called with two arguments, the pane itself, and one of the following keywords:

:tab-forward
    Tab was pressed.

:tab-backward
    Tab Backwards (usually Shift+Tab) was pressed.

:return
    Return was pressed.

:shift-return
    Shift+Return was pressed.

:enter
    Enter was pressed.

:shift-enter
    Shift+Enter was pressed.

Note: Enter is the key usually found on the numeric keypad.

When navigation-callback is non-nil, it is called instead of callback when Return is pressed. callback is still called via an OK button if there is one (see buttons below).

navigation-callback is implemented only on Microsoft Windows and Cocoa.

editing-callback, if non-nil, is a function of two arguments:

editing-callback pane type
pane is the text-input-pane and type is a keyword. editing-callback is called with type :start when the user starts editing and type :end when the user stops editing. In general, this occurs when the focus changes, but on Cocoa type :start is passed when the first change is made to the text.

gesture-callbacks provides callbacks to perform for specific keyboard gestures. Each gesture must be an object that sys:coerce-to-gesture-spec can coerce to a gesture-spec. Each callback can be a callable (symbol or function) which takes one argument, the pane. Alternatively each callback can be a list of the form (function arguments). Note that in this case, the pane itself is not automatically passed to the function amongst arguments.

When the user enters a gesture that matches gesture in any pair amongst gesture-callbacks, the callback is executed and the gesture is not processed any more.

**Note:** The interaction of in-place completion is implemented using gesture-callbacks. Gestures which you define explicitly by gesture-callbacks override the gestures which are defined implicitly by the in-place completion mechanism.

**Note:** For gestures that change the text, text-change-callback is probably better than gesture-callbacks.

When complete-do-action is non-nil, completion of the text in the pane automatically invokes callback (if callback is non-nil). The default value of complete-do-action is nil.

text-change-callback is a change callback (see change-callback) that is called only when the text in the pane changes. In contrast, change-callback is also called when the caret moves. If both text-change-callback and change-callback are supplied, only text-change-callback is invoked.
buttons specifies toolbar buttons which appear next to the pane and facilitate user actions on it. It also specifies the position of the buttons relative to the pane. This feature appears in the LispWorks IDE, for example the Class box of the Class Browser.

The allowed keys and values of the plist buttons are:

:ok  A boolean or a plist, default value t. If true, a button which calls callback appears. If the value is a plist then this plist supplies details for the button, as described below.

:cancel A boolean or a plist, default value nil. If true, a button which calls cancel-function appears. A plist value is interpreted as for :ok and can also contain the key :accelerator which specifies an accelerator used for the button. There is no default accelerator.

:completion A boolean or a plist. If true, a button which calls completion-function appears. The default value is t if completion-function is non-nil, and nil otherwise. A plist value is interpreted as for :ok.

:browse-file A keyword or a plist. If true, a button which invokes prompt-for-file appears. If the value is :save or :open then it is passed as the operation argument to prompt-for-file, replacing the text in the pane if successful. If the value is a plist, then it supplies details for the button, as described below, and can also contain the keywords :message to specify a message for the file prompter; :pathname to specify the default pathname of the file prompter (defaults to the text in the text-input-pane) or any of the keywords :ok-check, :filter, :fil-
ters, :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.

If search-field is a string and recent-items-name is not supplied, then the value search-field is used as the name. See the discussion of recent-items below.

If any of search-field, recent-items or recent-items-name is supplied and is non-nil, the pane uses NSSearchField, and also has "recent items". The NSSearchField has a different appearance from text-input-pane, can display recent items menu, and its input behavior is a little different too.

If recent-items is non-nil, it must be a list of strings, or t. When it is a list of strings, it specifies the initial list of "recent items". When it is t, it simply specifies that the pane should handle recent items.

If recent-items-name is non-nil, it should be a string. The string specifies the autosave name of the pane. When a pane has an autosave name, Cocoa remembers the list of recent items for pane with the same autosave name and same application. The record persists between invocations of the application.
If \textit{recent-items-name} is not supplied or is \texttt{nil}, and \textit{search-field} is a string, it is used instead as the name.

The maximum number of recent items defaults to 50 and can be controlled by the initarg value \texttt{maximum-recent-items}. The value 0 can be used to switch off the "recent items" feature, including the menu.

The recent items list can be read and set by \texttt{text-input-pane-recent-items}, or modified by any of \texttt{text-input-pane-replace-recent-items}, \texttt{text-input-pane-delete-recent-items}, \texttt{text-input-pane-append-recent-items}, \texttt{text-input-pane-prepend-recent-items} and \texttt{text-input-pane-set-recent-items}.

The input behavior of \texttt{text-input-pane} with "recent items" is the same is that of other \texttt{text-input-panes} except for the timing of calls to \textit{callback}. Note that this refers to the function that is passed with the initarg \texttt{:callback}. The \texttt{:change-callback} is not affected.

By default, each time the user types a character it causes a scheduling of \textit{callback} some short time later. If the user types another character before the callback, it is re-scheduled later. The result is that as long as the user types, there are no call-backs, but once the user stops a callback is generated.

The behavior of \textit{callback} can be controlled by the initarg value \texttt{recent-items-mode}, which can be one of \texttt{:explicit}, \texttt{:delayed} or \texttt{:immediate}. \texttt{:explicit} gives the same behavior as a normal \texttt{text-input-pane}, \texttt{:delayed} is the default described above, and \texttt{:immediate} means doing a callback immediately after each character. In addition, when the user selects an item from the recent items menu or clicks its \texttt{Cancel} button, the \textit{callback} is called. In the case of the \texttt{Cancel} button, the string would be empty.
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

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

(setq tip (capi:contain
  (make-instance
   'capi:text-input-pane
   :enabled nil)))

(capi:apply-in-pane-process
  tip #'(setf capi:text-input-pane-enabled) t tip)

(capi:apply-in-pane-process
  tip #'(setf capi:text-input-pane-enabled) nil tip)

(capi:apply-in-pane-process
  tip #'(setf capi:text-input-pane-text) "New text" tip)

(capi:contain (make-instance
  'capi:text-input-pane
  :text "Hello world"
  :callback #'(lambda (text interface)
                 (capi:display-message
                  "Interface ~S's text: ~S"
                  interface text)))))
```

This example uses a plist value for the `buttons` key :cancel to specify that the Cancel button is initially disabled:

```
(capi:contain
  (make-instance 'capi:text-input-pane
    :buttons
    '((ok t :cancel (:enabled nil)))))
```

This example shows how to specify a Help button which displays a help message:
(defvar *help-message* "A long help message."

(capi:contain
 (make-instance 'capi:text-input-pane
   :buttons
   "(:help
    (:title "help window"
     :message ,*help-message*)))))

This example illustrates the use of gesture-callbacks. Ctrl+e moves the cursor to the end of the input, Ctrl+a moves it to the start, and Ctrl+6 does something else:

(capi:contain
 (make-instance
   'capi:text-input-pane
   :gesture-callbacks
   (list
    (cons #\Ctrl-\e #'(lambda (tip)
      (setf (capi:text-input-pane-caret-position tip)
        (length (capi:text-input-pane-text tip))))
    (cons #\Ctrl-\a #'(lambda (tip)
      (setf (capi:text-input-pane-caret-position tip) 0)))
    (cons #\Ctrl-6 'do-something-else))))

There is a further example in the file examples/capi/elements/text-input-pane.lisp

See also
display-pane
editor-pane
multi-line-text-input-pane
text-input-choice
text-input-pane-complete-text
text-input-range
title-pane
**Function**

**text-input-pane-append-recent-items**

**Summary**
Modifies the recent items list in a `text-input-pane` on Cocoa.

**Signature**
`text-input-pane-append-recent-items text-input-pane &rest strings`

**Arguments**
- `text-input-pane` A `text-input-pane` with recent items.
- `strings` Strings.

**Values**
There is no meaningful return value.

**Description**
The function `text-input-pane-append-recent-items` modifies the recent items list in a `text-input-pane` that has recent-items (see `text-input-pane initargs :search-field, :recent-items and :recent-items-name`). It appends the strings at the end of the recent items, using `text-input-pane-set-recent-items` with `where = :end`.

**Notes**
`text-input-pane-append-recent-items` is implemented only on Cocoa.

**See also**
- `text-input-pane`
- `text-input-pane-set-recent-items`

**Function**

**text-input-pane-delete-recent-items**

**Summary**
Modifies the recent items list in a `text-input-pane` on Cocoa.

**Signature**
`text-input-pane-delete-recent-items text-input-pane &rest strings`

**Arguments**
- `text-input-pane` A `text-input-pane` with recent items.
- `strings` Strings.
Values
There is no meaningful return value.

Description
The function `text-input-pane-delete-recent-items` modifies the recent items list in a `text-input-pane` that has recent-items (see `text-input-pane` initargs `:search-field`, `:recent-items` and `:recent-items-name`). It deletes from the recent items any item that matches any of the strings (compared using `cl:string-equal`), using `text-input-pane-set-recent-items` with `where = :delete`.

Notes
`text-input-pane-delete-recent-items` is implemented only on Cocoa.

See also
`text-input-pane`
`text-input-pane-set-recent-items`

**text-input-pane-prepend-recent-items**  
*Function*

Summary
Modifies the recent items list in a `text-input-pane` on Cocoa.

Signature
`text-input-pane-prepend-recent-items` `text-input-pane` &rest `strings`

Arguments
`text-input-pane` A `text-input-pane` with recent items.
`strings` Strings.

Values
There is no meaningful return value.

Description
The function `text-input-pane-prepend-recent-items` modifies the recent items list in a `text-input-pane` that has recent-items (see `text-input-pane` initargs `:search-field`, `:recent-items` and `:recent-items-name`). It prepends the strings at the beginning of the recent items, using `text-input-pane-set-recent-items` with `where = :start`. 
text-input-pane-recent-items  

Summary  Gets and sets the recent items in a text-input-pane on Cocoa.

Signature  
text-input-pane-recent-items  text-input-pane => list-of-strings

(setf text-input-pane-recent-items)  list-of-strings  text-input-pane => list-of-strings

Arguments  
text-input-pane  A text-input-pane with recent items.
list-of-strings  A list of strings.

Description  The function text-input-pane-recent-items gets and sets the recent items in a text-input-pane that has recent-items. (see text-input-pane initargs :search-field, :recent-items and :recent-items-name).

The value list-of-strings passed to (setf text-input-pane-recent-items) must be a list of strings.

Notes  text-input-pane-recent-items is implemented only on Cocoa.

text-input-pane-recent-items does not work properly before the pane is displayed.

See also  text-input-pane
        text-input-pane-set-recent-items
text-input-pane-replace-recent-items

Function

Summary
Modifies the recent items list in a text-input-pane on Cocoa.

Signature
text-input-pane-replace-recent-items text-input-pane &rest strings

Arguments
text-input-pane A text-input-pane with recent items.
strings Strings.

Values
There is no meaningful return value.

Description
The function text-input-pane-replace-recent-items modifies the recent items list in a text-input-pane that has recent-items (see text-input-pane, initargs :search-field, :recent-items and :recent-items-name), using text-input-pane-set-recent-items with where = :replace.

text-input-pane-replace-recent-items replaces the recent items in the pane by the strings. It has the same effect as (setf text-input-pane-recent-items), but takes the strings as &rest arguments.

Notes
text-input-pane-replace-recent-items is implemented only on Cocoa.

See also
text-input-pane
text-input-pane-set-recent-items

text-input-pane-set-recent-items

Function

Summary
Sets the recent items in a text-input-pane.

Signature
text-input-pane-set-recent-items text-input-pane strings where
### Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>text-input-pane</code></td>
<td>A text-input-pane with recent items.</td>
</tr>
<tr>
<td><code>strings</code></td>
<td>A list of strings.</td>
</tr>
<tr>
<td><code>where</code></td>
<td>One of the keywords: <code>replace</code>, <code>delete</code>, <code>start</code> and <code>end</code>, or a non-negative integer.</td>
</tr>
</tbody>
</table>

### Values

The function `text-input-pane-set-recent-items` does not return a meaningful value.

### Description

The function `text-input-pane-set-recent-items` sets the recent items in a text-input-pane. The text-input-pane must have recent items, that is it must have been created with one of the keyword arguments: `search-field`, `recent-items` or `recent-items-name`. The `strings` argument must be a list of strings.

`text-input-pane-set-recent-items` modifies the recent items according to the argument `where`, which can one of:

- **replace**: The strings replace the recent items in the text-input-pane.
- **delete**: Delete from the recent items any item that matches any of the string (using cl:string-equal).
- **start**: Insert the strings at the beginning of the recent items.
- **end**: Insert the strings at the end of the recent items.

A non-negative integer

Insert the strings at the position indicated by the value. 0 means the same as `start`. If the integer is greater than the length of the current recent items list, the strings are inserted in the end of the list.
In all cases, if any of the strings is already in the recent-items list (as compared by \texttt{cl:string-equal}), it is first deleted from the list. This means that passing strings that already exist just moves them around in the list.

Notes \texttt{text-input-pane-set-recent-items} is a little more efficient than using \texttt{text-input-pane-recent-items} and \texttt{(setf text-input-pane-recent-items)} but the different is unlikely to be significant.

See also \texttt{text-input-pane} \texttt{text-input-pane-replace-recent-items} \texttt{text-input-pane-delete-recent-items} \texttt{text-input-pane-append-recent-items} \texttt{text-input-pane-prepend-recent-items} \texttt{text-input-pane-complete-text} \texttt{text-input-pane-replace-recent-items} \texttt{text-input-pane-delete-recent-items} \texttt{text-input-pane-append-recent-items} \texttt{text-input-pane-prepend-recent-items}

\textbf{Function} \\
\textbf{text-input-pane-complete-text} \\
\textbf{Summary} Calls the completion-function in a \texttt{text-input-pane}. \\
\textbf{Package} capi \\
\textbf{Signature} \texttt{text-input-pane-complete-text} \texttt{pane} $\Rightarrow$ \texttt{result} \\
\textbf{Arguments} \texttt{pane} A \texttt{text-input-pane}. \\
\textbf{Values} \texttt{result} A string, or \texttt{nil}. \\
\textbf{Description} The function \texttt{text-input-pane-complete-text} calls the completion-function of \texttt{pane} with the current \texttt{text}. If this call is successful, then the \texttt{text} of \texttt{pane} is set to the result, and \texttt{text-input-pane-complete-text} returns this result. Otherwise, \texttt{result} is \texttt{nil}. \\
\textbf{Note:} the completion-function may return a list of completion candidates, in which case \texttt{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.
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 `t` if there is a selected region in `pane` and `nil` otherwise.

See also

`set-text-input-pane-selection`  
`text-input-pane`  
`text-input-pane-selected-text`  
`text-input-pane-selection`
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.
- `:value` An integer specifying the current value in the pane.
- `:callback` A function called when the value is changed by the user.
- `:change-callback` A function called called when the user edits the text in the pane.
- `: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-change-callback
text-input-range-callback-type
Description

The class **text-input-range** provides numeric input of integers in a given range (some systems refer to this 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**, if non-nil, should be a function to be called whenever the value is changed by the user. The arguments to **callback** 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)**. Note that, if the value is changed by the user editing the text, then **change-callback**, if supplied, is called as well.

**change-callback**, if non-nil, should be a function of four arguments, to be called when the user edits the text in the pane. It should have this signature:

```
callback string pane interface caret-position
```

where the arguments are interpreted just as for the change-callback of **text-input-pane**. Note that editing of the text may or may not change the value in the **text-input-range** (that is, what **text-input-range-value** returns). If the value does change, then **callback** is called too.

Notes

On Cocoa, **change-callback** is not called for a cursor move only.

Example

```
(capi:contain
 (make-instance 'capi:text-input-range
   :start 0
   :end 100
   :value 42))
```
class title-pane

summary
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
titled-menu-object
text-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-call-back-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

```lisp
(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
  (make-instance 'capi:list-panel
    :items '(1 2 3 4 5)
    :title "Temp"
    :title-position :left
    :title-adjust :center
    :title-args
      '(:visible-min-width (:character 12))))

mnemonic-title offers an alternate way to provide the pane’s title, and with a mnemonic. It takes effect only for button-panel, list-panel, list-view, option-pane, output-pane, progress-bar, scroll-bar, slider, text-input-pane, text-input-range, tree-view and their subclasses, and is interpreted as described for menu.

Note: titles and mnemonic titles can now be added in a grid-layout.
titled-object corresponds to the LispWorks 4.1 class titled-pane. For backwards compatibility the accessors titled-pane-title and titled-pane-message, including setf methods, are provided. These simply trampoline to titled-object-title and titled-object-message, and may not be supported in future releases.

Example

Try each of these examples to see some of the effects that titled panes can produce. Note that text-input-pane is a subclass of titled-object, and that it has a default title-position of :left.

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

(capi:contain (make-instance 'capi:text-input-pane :title "Enter some text:"))

(capi:contain (make-instance 'capi:text-input-pane :title "Enter some text:" :title-position :top))


(capi:contain (make-instance 'capi:text-input-pane :message "A message"))

(capi:contain (make-instance 'capi:text-input-pane :message "A message" :title "Enter some text:"))

(capi:contain (make-instance 'capi:text-input-pane :title "Enter some text:" :title-args '((:foreground :red))))

See also message-pane title-pane
### titled-pinboard-object

**Class**

**Summary**
A pinboard object with a title.

**Package**
capi

**Superclasses**
- pinboard-object
- titled-object

**Subclasses**
- image-pinboard-object

**Description**
The class `titled-pinboard-object` provides a pinboard object with a title. The title is regarded as part of the object in geometry calculations.

**Note:** `titled-pinboard-object` does not allow the value `:frame` for the `titled-object` initarg `title-position`. The values `:top`, `:bottom`, `:left` and `:right` are allowed.

**Example**
This example creates three instances of `titled-pinboard-object` and one of `item-pinboard-object`, all with 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
Class

toolbar

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

Summary
This class is used to create instances of toolbar buttons.

Package
capi

Superclasses
item
toolbar-object

Initargs
:callback
A function that is called when the user presses the toolbar button and popup-interface is non-nil.

:image
Specifies the image to use for the toolbar button.

:selected-image
Specifies the image to use for the toolbar button when it is selected.

:tooltip
An optional string which is displayed, on Microsoft Windows, when the mouse moves over the button. :tooltip is deprecated.
:help-key   An object used for lookup of help. Default value t.
:remapped   Links the button to a menu item.
:dropdown-menu
   A menu or nil.
:dropdown-menu-function
   A function of no arguments, or nil.
:dropdown-menu-kind
   One of the keywords :button, :only and :delayed.
:popup-interface
   An interface or nil.

Accessors toolbar-button-image
toolbar-button-selected-image
toolbar-button-dropdown-menu
toolbar-button-dropdown-menu-function
toolbar-button-dropdown-menu-kind
toolbar-button-popup-interface

Readers help-key

Description Toolbar buttons may be placed within toolbars and toolbar components. However, there is usually no need to create toolbar buttons explicitly; instead, the callbacks and images arguments to toolbar or toolbar-component can be used. To add tooltips, use the interface help-callback with help-key :tooltip.

In addition, an interface can have its own toolbar buttons, specified by its toolbar-items. There is no toolbar object in that situation.

image and selected-image may each be one of the following:
A pathname or string

This specifies the filename of a file suitable for loading with \texttt{load-image}. Currently this must be a bitmap file.

A symbol

The symbol must either have been previously registered by means of a call to \texttt{register-image-translation}, or be one of the following symbols, which map to standard images: \texttt{:std-cut}, \texttt{:std-copy}, \texttt{:std-paste}, \texttt{:std-undo}, \texttt{:std-redo}, \texttt{:std-delete}, \texttt{:std-file-new}, \texttt{:std-file-open}, \texttt{:std-file-save}, \texttt{:std-print}, \texttt{:std-print-pre}, \texttt{:std-properties}, \texttt{:std-help}, \texttt{:std-find} and \texttt{:std-replace}

On Microsoft Windows, the following symbols are also recognized for view images: \texttt{:view-large-icons}, \texttt{:view-small-icons}, \texttt{:view-list}, \texttt{:view-details}, \texttt{:view-sort-name}, \texttt{:view-sort-size}, \texttt{:view-sort-date}, \texttt{:view-sort-type}, \texttt{:view-parent-folder}, \texttt{:view-net-connect}, \texttt{:view-net-disconnect} and \texttt{:view-new-folder}.

Also on Microsoft Windows, these symbols are recognized for history images: \texttt{:hist-back}, \texttt{:hist-forward}, \texttt{:hist-favorites}, \texttt{:hist-addtofavorites} and \texttt{:hist-viewtree}.

An image object, as returned by \texttt{load-image}.

An image locator object

This allows 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, 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 the toolbar-button as its single argument. It 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 (ignorable data interface))
  (capi:display-message "Doing Redo"))
```

A simple interface:
In this interface, pressing the toolbar button invokes the menu item callback:

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

This last example illustrates the use of :selected-image.
toolbar-component

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.

:selection-function
  A function to dynamically compute the selection.

:selected-item-function
  A function to dynamically compute the selected item.

:selected-items-function
  A function to dynamically compute the selected items.

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`.

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 is non-nil, it should be a function which is called before the `toolbar-component` is displayed and when `update-toolbar` is called and which determines which items are selected. The function takes a single argument, which is the `interface` of the `toolbar-component`.

`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 `toolbar-component`, or is equal to such an item when compared by the `toolbar-component`'s `test-function` and `key-function`.

`selected-items-function`, if non-nil, should return a list of such objects.

**Example**

See `examples/capi/elements(toolbar.lisp)`.

**See also**

`toolbar`

`toolbar-button`

**toolbar-object**

**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.

Notes The function enabled-function should not display a dialog or do anything that may cause the system to hang. In general this means interacting with anything outside the Lisp image, including files, databases and so on.

See also toolbar
toolbar-button
toolbar-component
update-toolbar

top-level-interface Generic Function

Summary Returns the top level interface containing a specified pane.

Package capi
1  CAPI Reference Entries

**Signature**  top-level-interface  pane

**Description**  Returns the top level interface that contains pane.

**See also**  top-level-interface-p

interface
element

top-level-interface-display-state

*Generic Function*

**Summary**  Returns a value which indicates how the top level interface is displayed.

**Package**  capi

**Signature**  top-level-interface-display-state  interface

**Arguments**  interface  A top level interface or dialog window

**Description**  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 top-level-interface-display-state returns a value that indicates the current state of the interface interface. The following values can be returned:

: 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 => tx, ty, twidth, theight

**Arguments**

interface An interface.

**Values**

* tx, ty, twidth, theight
  Integers.

**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.

* tx and ty are measured from the top-left of the screen rectangle representing the area of the primary monitor (the primary screen rectangle).
Notes

On Cocoa, the result does not account for the size of the interface toolbar, if present in 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
<table>
<thead>
<tr>
<th>Arguments</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>interface</td>
<td>key</td>
</tr>
<tr>
<td></td>
<td>product-name</td>
</tr>
</tbody>
</table>

Description
The generic function `top-level-interface-geometry-key` returns as multiple values a key and a product name, which determine where the geometry of `interface` is saved. The saved geometry is used when displaying a future instance.

The supplied method on `interface` returns the class name of `interface` as the `key`, and `nil` as the `product-name`. You can define methods for your interfaces and products.

- `key` must be a symbol.
- `product-name` is used to derive the `product-registry-path`.
- `product-name` can be a symbol which was previously defined to have a registry path by
  `(setf sys:product-registry-path)`.
- `product-name` can alternatively be a string, which is taken directly as `product-registry-path`.
- `product-name` can alternatively be a list of strings, denoting multiple path components. These are concatenated together with the appropriate separator for the platform to give `product-registry-path`.

The geometry of `interface` is saved at the path which is constructed by concatenating (with appropriate separators) these values:

```
user-path product-registry-path "Environment" (symbol-package key) (symbol-name key)
```

where `user-path` is the registry branch
HKEY_CURRENT_USER on Microsoft Windows and the home directory on Unix/Linux and Mac OS X.
Note: for your interface classes for which you want the geometry to be saved, define a method on `top-level-interface-save-geometry-p`.

Note: in an image delivered at delivery level 5, symbol names are removed by default. This breaks the saved geometry mechanism as the registry path is constructed using `symbol-name`. To make this work in a level 5 delivered image, explicitly keep the `key` symbol. See the *LispWorks Delivery User Guide* for details.

See also `top-level-interface-save-geometry-p`

---

**top-level-interface-p**

*Generic Function*

**Summary**
The predicate for top level interfaces.

**Package**
capi

**Signature**
top-level-interface-p pane

**Description**
The generic function `top-level-interface-p` returns true if `pane` is a top level interface.

See also `top-level-interface`

**top-level-interface-save-geometry-p**

*Generic Function*

**Package**
capi

**Signature**
top-level-interface-save-geometry-p interface => result
### Description
The generic function `top-level-interface-save-geometry-p` returns true if the geometry of `interface` should be saved for use by a future instance.

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

**See also** `top-level-interface-geometry-key`

---

### tracking-pinboard-layout

**Class**

**Summary**
A pinboard with automatic highlighting.

**Package**
capi

**Superclasses**
pinboard-layout

**Description**
The class `tracking-pinboard-layout` provides a pinboard which tracks mouse movement by highlighting its objects as the mouse cursor moves over them.

This functionality is implemented via a `:motion` specification in the `input-model`. Therefore, you may not specify `:motion` in the `input-model` of a `tracking-pinboard-layout`. See `output-pane` for a description of `input-model`. 
Example

(defclass my-ellipse (capi:drawn-pinboard-object)
  ((color :initarg :color
           :initform :red
           :accessor my-ellipse-color)))

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

(defun change-ellipse-color (pinboard x y)
  (let ((ellipse
          (capi:pinboard-object-at-position
           pinboard x y)))
    (when ellipse
      (let ((color
              (capi:prompt-for-color
               "New color"
               :color
               (my-ellipse-color ellipse)
               :owner
               (capi:convert-to-screen))))
        (when color
          (setf (my-ellipse-color ellipse) color)
          (capi:with-geometry ellipse
           (gp:invalidate-rectangle
            pinboard
            capi:%x%
            capi:%y%
            capi:%width%
            capi:%height%))))))

(capi:contain
  (make-instance
   'capi:tracking-pinboard-layout
   :description
   (loop for i below 20
     collect
     (make-instance 'my-ellipse
       :x (+ 5 (random 290))
       :y (+ 5 (random 290)))))

638
tree-view

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.

: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.
Initargs for handling check boxes. Note that these do not work on Cocoa:

:checkbox-status
Controls whether the tree has checkboxes, except on Cocoa. 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.

Initargs for handling images:
: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.

: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.
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.

expandp-function controls automatic expansion of nodes (items) in the tree-view. By default, initially only the items specified by the roots argument are displayed. This initial display can be altered by supplying a function expandp-function which allows further items to be displayed. If supplied, expandp-function should be a function which is called on the roots and is called recursively on the children if it returns true. When the user expands a node, expandp-function is called on each newly created child node, which is expanded if this call returns true, and so on recursively. The default value of expandp-function is nil so that there is no automatic expansion and only the root nodes are visible initially.

The default value of retain-expanded-nodes is t.
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`.

`delete-item-callback` is called when the user presses the `Delete` key. Two arguments are passed: the `tree-view` and the selected item `item`. Note that, apart from calling the callback, the system does nothing in response to the `Delete` key. In particular, if you want to remove the selected `item`, `delete-item-callback` needs to do it by changing what the `children-function` returns when called on the parent of `item`. Normally you also need to to call `tree-view-update-item` with `in-parent = t` to actually update the tree on the screen.

Note also that in `extended-selection-tree-view` (a subclass of `tree-view`), if the `interaction` was not explicitly changed to `:single-selection`, the second argument to `delete-item-callback` is a list of the selected items (even when only one item is selected).

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`. It can also one of the following symbols, which map to standard images: `:std-cut`, `:std-copy`, `:std-paste`, `:std-undo`, `:std-redo`, `:std-delete`,

643
:std-file-new, :std-file-open, 
:std-file-save, :std-print, 
:std-print-pre, :std-properties, 

On Microsoft Windows, the following symbols are also recognized. They map to view images: :

Also on Microsoft Windows, these symbols are recognized. They map to history images: :

An image object, as returned by load-image.

An image locator object

This allowing a single bitmap to be created which contains several button images side by side. See make-image-locator for more information. On Microsoft Windows, it also allows access to bitmaps stored as resources in a DLL.

An integer

This is a zero-based index into the tree-view’s image lists. 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 objects listed above,
just as for image-function, or nil to indicate that there is no
state image. See also checkbox-status, which overrides the
state-image-function.

If image-lists is specified, it should be a plist containing the
following keywords as keys. The corresponding values
should be image-list objects.

: normal Specifies an image-list object that contains
the item images. The image-function should
return a numeric index into this image-list.

: state Specifies an image-list object that contains
the state images. The state-image-function
should return a numeric index into this
image-list.

If right-click-extended-match is nil, the mouse right button
gesture within the tree view selects an item only when the
cursor is on the item. Otherwise, this gesture also selects an
item to the left or right of the cursor. The default for right-
click-extended-match is t.

If has-root-line is nil, the vertical root line and expanding
boxes of the root nodes are not drawn. This is useful in two
cases:

• When the tree view needs to be neater. Note that the user
does not have a mouse gesture to expand the root node.
Normally the programmer would compensate for this by
making some other gesture call (setf tree-view-
expanded-p).

• If a children-function is not supplied, this can be used to
create a pane like a list view with checkboxes (see below
for details of checkboxes). This pane can be handled as if
it is a typical choice, except that setting the items is done
by (setf tree-view-roots) or by passing :roots to
make-instance. In a typical choice, you would do
(setf collection-items) or pass :items to
make-instance.

The default for has-root-line is t.

If the checkbox-status is non-nil then the tree view provides an
automatic way of using the state images as checkboxes
(except on Cocoa where check boxes are not supported). The
state-image is defaulted to a set of images containing check-
boxes and the state-image-function is ignored, but each item
has a status that is a non-negative integer no greater than the
number of images in state-image-list. The status specifies
which image is displayed alongside item.

When item is expanded in the tree for the first time, the status
of each child is set to item’s status. The status can be changed
interactively by the user:

- Left mouse button on a checkbox changes its status
- Space changes the status of all selected items.

The status can also be read and set programmatically (see
tree-view-item-checkbox-status).

When the status of an item changes:

- The statuses of its ancestors may change if a checkbox-
  parent-function was supplied.
- The statuses of an items descendents may change if a
  checkbox-child-function was supplied.
- A callback given by checkbox-callback-function will be
called, if this was supplied.

By default checkboxes have three statuses indicated by
images: un-checked(0), grey-checked(1) and checked(2). If an
item is checked or un-checked, then all its descendents have
the same status. If an item is grey-checked, then its descen-
Students 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. Note that check boxes are not supported on Cocoa.

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.

3. On Microsoft Windows, the system always sets the input focus to the tree-view after its selection-callback returns. If you need this callback to set the focus elsewhere, call set-pane-focus outside the callback, like this:

   (mp:process-send process (list 'capi:set-pane-focus pane))

See also
choice
extended-selection-tree-view
tree-view-ensure-visible
tree-view-expanded-p
tree-view-item-checkbox-status
tree-view-item-children-checkbox-status
tree-view-update-item

tree-view-ensure-visible

Function

Summary
Ensures that an item in a tree-view is visible.

Package
capi

Signature
tree-view-ensure-visible tree-view item

Arguments


Description

The function `tree-view-ensure-visible` ensures that an item in a tree view is visible, scrolling the tree view if necessary.

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**

- `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`.

---

1  CA PI Reference Entries
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, except on Cocoa.

(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, except on Cocoa.

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

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>tree-view</code></td>
<td>A tree-view.</td>
</tr>
<tr>
<td><code>item</code></td>
<td>An item.</td>
</tr>
<tr>
<td><code>in-parent</code></td>
<td>A boolean.</td>
</tr>
</tbody>
</table>

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*

<table>
<thead>
<tr>
<th>Package</th>
<th>capi</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Signature</th>
<th><code>undefine-menu function-name &amp;rest args</code></th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Description</th>
<th>This function undefines a menu created with <code>define-menu</code>.</th>
</tr>
</thead>
</table>

| See also | `define-menu` menu |

**unhighlight-pinboard-object**

*Generic Function*

<table>
<thead>
<tr>
<th>Summary</th>
<th>Removes the highlighting from a <code>pinboard-object</code>.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Package</th>
<th>capi</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Signature</th>
<th><code>unhighlight-pinboard-object pinboard object &amp;key redisplay</code></th>
</tr>
</thead>
</table>
**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`.

**See also** toolbar
toolbar-button
toolbar-component

**virtual-screen-geometry**

*Function*

**Summary** Returns, as multiple values, a screen rectangle covering the full area of all the monitors associated with a screen.

**Package** capi

**Signature** `virtual-screen-geometry screen => x, y, width, height`

**Arguments**
- `screen` A CAPI screen.

**Values**
- `x` An integer.
- `y` An integer.
width A positive integer.
height A positive integer.

Description The function virtual-screen-geometry returns the "virtual" geometry of the screen screen, which is a screen rectangle covering the full area of all the monitors that are associated with screen.

The screen rectangle is at coordinates x and y as offsets from the top-left of the primary screen, with dimensions width and height.

See also pane-screen-internal-geometry
screen-internal-geometries
screen-monitor-geometries

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.

Notes with-atomic-redisplay does not cause Graphics Ports drawing operations to the panes to be deferred.
with-busy-interface

**Macro**

**Summary**
Displays an alternate cursor during the execution of some code, on platforms other than Cocoa.

**Package**
capi

**Signature**
with-busy-interface (pane &key cursor delay) &body body

**Description**
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

\[
\text{with-dialog-results (\&rest results) dialog-form &body body} \\
\Rightarrow :\text{continuation}, \text{nil}
\]

Arguments

- \text{results} Variables.
- \text{dialog-form} A function call form.
- \text{body} Forms.

Description The macro \text{with-dialog-results} is designed to evaluate the \text{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 \text{dialog-form} should be a function call form that displays a dialog.

The overall effect is that the \text{body} forms are evaluated with the \text{results} variables bound to the values returned by the \text{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 \text{with-dialog-results} macro waits until the dialog has been dismissed and then evaluates the \text{body} forms.

- On Cocoa, the \text{dialog-form} creates a sheet attached to the active window and the \text{with-dialog-results} macro returns immediately. The \text{body} forms are evaluated when the user dismisses the sheet.

The \text{dialog-form} must be a cons with one of the following two formats:

- \text{(function-name . arguments)}
((apply function-name . arguments))

The function-name is called with all the given arguments, plus an additional pair of arguments, :continuation and a continuation function created from body. In the first format, the additional arguments are placed after all the given arguments. In the second format, the additional arguments are placed just before the last of the given arguments (i.e. before the list of remaining argument to apply).

The continuation function binds the results variables to its arguments and evaluates the body forms. If there are more arguments than results variables, the extra arguments are discarded.

This macro is designed for use with function-names such as popup-confirm or prompt-for-string, which take a :continuation keyword. You can define your own such functions provided that they call one of the CAPI functions, passing the received continuation argument.

Examples

On Microsoft Windows, GTK+ and Motif, this displays a dialog, calls record-label-in-database when the user clicks OK and then returns. On Cocoa, this creates a sheet and returns; record-label-in-database will be called when the user clicks OK.

(with-dialog-results (new-label okp)
  (prompt-for-string "Enter a label")
  (when okp ; the user clicked in the OK button
    (record-label-in-database new-label)))

Here is an example with skeleton code for using with-dialog-results. Note that the dialog function (choose-file below) that is called by with-dialog-results must take a continuation keyword argument and pass it to a CAPI prompting function. Also note that the call to the CAPI prompting function must be the last form in the dialog func-
Forms after the CAPI prompting function will be executed at an indeterminate time, and their values will not be used in the body of `with-dialog-results`

```lisp
(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`

**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
with-document-pages

**Signature**

\[
\text{with-document-pages page-var first-page last-page &body body}
\]

**Description**

The `with-document-pages` evaluates `body` repeatedly, with `page-var` bound to the number of the page to print on each iteration. It is used to by applications providing Page on Demand printing.

The `first-page` and `last-page` arguments are evaluated to yield the page numbers of the first and last pages in the document.

`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**

\[
\text{with-external-metafile (var &key pane bounds format pathname drawing-mode) &body body => nil}
\]

**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 the keywords `:enhanced`, `:enhanced-plus`, `:enhanced-gdi` and `:windows`.

- `pathname`  
  A pathname or string.
drawing-mode  One of the keywords :compatible and :quality.

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 and GTK+ the metafile format is PDF.

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.

format is used only on Microsoft Windows. It can be one of:

:enhanced  Generate an Enhanced-metafile file containing "dual drawing" both in GDI+ and GDI.

:enhanced-plus  Generate an Enhanced-metafile file containing drawing only in GDI+.
:enhanced-gdi Generate an Enhanced-metafile file containing drawing only in GDI.

:windows Generate a Windows-metafile.

The default value of format is :enhanced.

When drawing-mode is :compatible (rather than the default value :quality) :enhanced and :enhanced-plus behave like :enhanced-gdi.

Note: GDI+ gives the best quality, so normally that is what you would want. However some programs may be able to display only GDI (and not GDI+), which is why the default is dual drawing. This however generates a larger file and is presumably slightly slower, so if you are sure that the file will be used only by programs that can draw GDI+ emf files (sometimes called EMF+), you can use format :enhanced-plus.

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.

drawing-mode should be either :compatible which causes drawing to be the same as in LispWorks 6.0, or :quality which causes all the drawing to be transformed properly, and allows control over anti-aliasing on Microsoft Windows and GTK+. The default value of drawing-mode is :quality.

For more information about drawing-mode, see "Drawing mode and anti-aliasing" in the CAPI User Guide.

Notes

with-external-metafile is not implemented on X11/Motif.

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`. Except the special cases which are mentioned below, these variables are read-only and should not be set.

Four variables define the geometry of the pane. If you define your own `calculate-layout` method, it can set these variables:

- `%x%` An integer specifying the x position of the pane in pixels relative to its parent.
- `%y%` An integer specifying the y position of the pane in pixels relative to its parent.
- `%width%` An integer specifying the width in pixels of the pane.
- `%height%` An integer specifying the height in pixels of the pane.

Four variables specify constraints on the pane. If you define your own `calculate-constraints` method, it can set these variables:

- `%min-width%` A real number specifying the minimum width of the pane.
A real number specifying the minimum height of the pane.

A real number specifying the maximum width of the pane.

A real number specifying 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

The following two variables access the object for which the representation is:
%object%
   The object whose geometry this is.
%child%
   The same as %object% (kept for compatibility with LispWorks 3.1).

See also calculate-constraints
        calculate-layout
        convert-relative-position
        element
        get-horizontal-scroll-parameters
        get-vertical-scroll-parameters
        scroll
        set-horizontal-scroll-parameters
        set-vertical-scroll-parameters

with-internal-metafile
   Macro

Summary  Creates a metafile in memory using Graphics Ports operations.

Package   capi

Signature with-internal-metafile (var &key pane bounds format
drawing-mode) &body body => metafile

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 the keywords :enhanced, :enhanced-plus and :enhanced-gdi.

drawing-mode One of the keywords :compatible and :quality.

body Lisp code.

Values metafile A metafile.

Description The macro with-internal-metafile creates a metafile containing records corresponding to the Graphics Ports operations in body that draw to var.

with-internal-metafile behaves like with-external-metafile except that an object representing the metafile is returned, and no file is created on disk.

var, pane, bounds, format, drawing-mode and body are interpreted as for with-external-metafile except that format cannot have the value :windows.

Note: GDI+ gives the best quality, so normally that what you want. But you cannot put a GDI+ only metafile on the clipboard, which is why the default is to make a "dual" metafile containing both GDI and GDI+ drawing. If are not going to put the metafile on the clipboard (by calling set-clipboard with format :metafile) you can use format :enhanced-plus which is slightly faster and uses less memory.

metafile must be freed after use, by calling free-metafile.

Notes 1. with-internal-metafile is supported on GTK+ only where Cairo is supported (GTK+ version 2.8 and later).

2. On GTK+, the internal metafile is slow to resize, so it is probably not useful when it is frequently resized (that is, drawn with different width or height).
3. **with-internal-metafile** is not implemented on X11/Motif.

Examples

```lisp
examples/capi/graphics/metafile.lisp
examples/capi/graphics/metafile-rotation.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**

```lisp
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 \texttt{interactive} is \texttt{t} then \texttt{print-dialog} is called to select the
printer to use. If \texttt{interactive} is \texttt{nil} then \texttt{printer} is used unless it
is \texttt{nil} in which case the \texttt{current-printer} is used. The
default value of \texttt{interactive} is \texttt{t} and the default value of \texttt{printer}
is \texttt{nil}.

The values of \texttt{jobname} and \texttt{tab-spacing} are passed to \texttt{print-text},
which is used to actually do the printing. The default
value of \texttt{tab-spacing} is 8 and the default value of \texttt{jobname} is
"Text".

\begin{itemize}
\item \texttt{current-printer}
\item \texttt{print-dialog}
\item \texttt{print-text}
\end{itemize}

\begin{description}
\item[with-page] \textbf{Macro}
\item[Summary] Binds a variable to either \texttt{t} or \texttt{nil}, and executes a body of
code to print a page only if the variable is \texttt{t}.
\item[Package] \texttt{capi}
\item[Signature] \texttt{with-page \ (printp) \ &body \ body}
\item[Description] The \texttt{with-page} macro binds \texttt{printp} to \texttt{t} if a page is to be
printed, or \texttt{nil} if it is to be skipped. The \texttt{body} is executed
once, and is expected to draw the document only if \texttt{printp} is
\texttt{t}.

Each call to \texttt{with-page} contributes a new page to the docu-
ment.
\item[Note:] \texttt{with-page} does not work on Cocoa.
\item[See also] \texttt{with-document-pages}
\item[See also] \texttt{with-page-transform}
\end{description}
**with-page-transform**

*Macro*

**Summary**
Defines a rectangular region within the coordinate space of an output pane or printer port.

**Package**
capi

**Signature**
```lisp
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**
```lisp
with-print-job (var &key pane jobname printer drawing-mode) &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.

`drawing-mode` should be either `:compatible` which causes drawing to be the same as in LispWorks 6.0, or `:quality` which causes all the drawing to be transformed properly, and allows control over anti-aliasing on Microsoft Windows and GTK+. The default value of `drawing-mode` is `:quality`.

For more information about `drawing-mode`, see "Drawing mode and anti-aliasing" in the CAPI User Guide.

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`
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** `wrap-text text width &key start end => strings`

**Arguments**

- `text` A string.
- `width` A positive integer.
- `start`, `end` Bounding index designators of `text`.

**Values**

- `strings` A list of strings.

**Description** The function `wrap-text` takes a string `text` and returns a list of strings, each of which is no longer than `width`. Together the strings in `strings` contain all the non-whitespace characters of `text` between `start` and `end` and are suitable for displaying this text on multiple lines of length `width`.

See also `wrap-text-for-pane`
Function

**wrap-text-for-pane**

Summary  Wraps text for a given pane.

Package  capi

Signature  

```
wrap-text-for-pane pane text &key external-width visible-width
font start end => strings
```

Arguments  

- **text**: A string.
- **pane**: A displayed CAPI pane.
- **external-width**: An integer or `nil`.
- **visible-width**: An integer or `nil`.
- **font**: A font object.
- **start**: An integer.
- **end**: An integer or `nil`.

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 x direction.
:y-adjust The adjust value for the y direction.

Accessors
layout-x-adjust
layout-y-adjust

Description
The values x-adjust and 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 (keyword n) where n is an integer. These values of adjust are interpreted as by pane-adjusted-position.

:top is the default for y-adjust and :left is the default for x-adjust.
Example

Note: `column-layout` is a subclass of `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`
2pi

Summary: (* 2 pi) as a double-float.

Package: graphics-ports

Description: The constant 2pi is the result of (* 2 cl:pi). It is a cl:double-float.

See also: fpi, pi-by-2
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 rotation of a given number of radians is performed on any points multiplied by the transform.

Package
graphics-ports

Signature
apply-rotation transform theta => transform

Arguments
transform A transform.
theta A real number.
The function **apply-rotation** 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.

The rotation is around the point (0,0).

If *theta* is positive, then the rotation is clockwise.

**apply-rotation** returns the transform.

**Examples**

`examples/capi/graphics/metafile-rotation.lisp`

**See also**

`apply-rotation-around-point`

`apply-scale`

`apply-translation`

---

**apply-rotation-around-point**

**Function**

**Summary**

Modifies a transform such that a specified rotation around a specified point is performed on any points multiplied by the transform.

**Package**

*graphics-ports*

**Signature**

`apply-rotation-around-point transform theta x y => transform`

**Arguments**

- `transform`: A transform.
- `theta`: A real number.
- `x`: A real number.
- `y`: A real number.
Description
The function \texttt{apply-rotation-around-point} modifies \texttt{transform} such that a clockwise rotation of \texttt{theta} radians around the point \((x,y)\) is performed on any points multiplied by the transform. Any operations already contained in the transform occur before the new rotation.

\texttt{apply-rotation-around-point} returns the transform.

Examples
\begin{verbatim}
examples/capi/graphics/rotation-around-point.lisp
\end{verbatim}

See also
\texttt{apply-rotation}

\begin{description}
\item[Function] \texttt{apply-scale}
\item[Summary] Modifies a transform such that a scaling occurs on any points multiplied by the transform.
\item[Package] \texttt{graphics-ports}
\item[Signature] \texttt{apply-scale \ transform \ sx \ sy \ => \ transform}
\item[Arguments]
\begin{itemize}
\item[\texttt{transform}] A transform.
\item[\texttt{sx}] A real number.
\item[\texttt{sy}] A real number.
\end{itemize}
\item[Description] The function \texttt{apply-scale} modifies \texttt{transform} such that a scaling of \texttt{sx} in \texttt{x} and \texttt{sy} in \texttt{y} is performed on any points multiplied by the transform. Any operations already contained in the transform occur before the new scaling.

\texttt{apply-scale} returns the transform.
\item[Examples]
\begin{verbatim}
examples/capi/graphics/metafile-rotation.lisp
\end{verbatim}
\end{description}
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 => transform`

**Arguments**
- `transform`: A transform.
- `dx`: A real number.
- `dy`: A real number.

**Description**
The function `apply-translation` 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.

`apply-translation` returns the transform.

**Examples**
`examples/capi/graphics/metafile-rotation.lisp`

**See also**
- `apply-rotation`
- `apply-rotation-around-point`
- `apply-scale`

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

Signature  augment-font-description  fdesc  &rest  font-attribute*  =>  return

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  Function

Summary  Clears external image conversions for a port.

Package  graphics-ports

Signature  clear-external-image-conversions  external-image  gp-or-null
  &key  free-image  all  errorp

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.

clear-rectangle is deprecated.

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 deprecated function clear-rectangle draws the rectangle specified by x, y, width, and height in port’s background color. All other graphics-state parameters are ignored.

clear-rectangle is deprecated because it ignores the graphics state args, which means it does not work properly with other drawing functions. In particular, it does not work properly in the display-callback of output-pane.

Use instead:

(draw-rectangle pane x y width height
 :filled t
 :foreground color
 :compositing-mode :copy
 :shape-mode :plain)

compositing-mode is needed only when the color has alpha.
foreground is needed only if it is different from the foreground in the graphics state.

Note that draw-rectangle does take into account the transformation in the graphics-state.

See also draw-rectangle

compress-external-image

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

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

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>port</td>
<td>A CAPI pane.</td>
</tr>
<tr>
<td>string</td>
<td>A string.</td>
</tr>
<tr>
<td>font</td>
<td>A font.</td>
</tr>
</tbody>
</table>

Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>extents</td>
<td>An array of integers.</td>
</tr>
</tbody>
</table>

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*

**Signature**

```
convert-external-image gp external-image &key cache force-new => image
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gp</td>
<td>A CAPI pane.</td>
</tr>
<tr>
<td>external-image</td>
<td>An external image.</td>
</tr>
<tr>
<td>cache</td>
<td>A boolean.</td>
</tr>
<tr>
<td>force-new</td>
<td>A boolean.</td>
</tr>
</tbody>
</table>

**Values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>image</td>
<td>An image.</td>
</tr>
</tbody>
</table>
**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-area**

*Function*

**Summary**
Copies a rectangular area from one port to another.

**Package**
*graphics-ports*

**Signature**

```
copy-area to-port from-port to-x to-y width height from-x from-y &rest args
```

**Arguments**

- `to-port` A graphics port.
- `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.
- `args` *graphics-state* parameters passed as keyword arguments.

**Description**

The function *copy-area* copies a rectangular area from one port to another, taking account of transformations.

In *drawing-mode:* :compatible (old drawing mode), *copy-area* does exactly the same as *copy-pixels*.

In *drawing-mode:* :quality (the default) it copies a rectangular area from one port to another. The *transform, mask, mask-transform, compositing-mode* and *shape-mode* of *to-port's* *graphics-state* are used. The *to-port* and *from-port* need not be the same depth. They can be the same object. The *from-x* and *from-y* values are interpreted as pixel positions in the window coordinates of *from-port*, that is, they are not transformed by *from-port’s* transform.
Notes

The main difference between `copy-area` and `copy-pixels` in `drawing-mode :quality` is when copying from a displayed window.

`copy-area` always copies using the right transformation of the target, but it means that it may copy from an obscured part of the window and hence copy the wrong thing. `copy-pixels` generates an exposure event on the target port instead of copying obscure areas, but to do that it has to ignore the transformation.

Examples

tables/capi/graphics/compositing-mode.lisp

See also

copy-pixels
graphics-state

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**  
\[ \text{copy-pixels to-port from-port to-x to-y width height from-x from-y &rest args} \]

**Arguments**  
- `to-port`  
  A graphics port.
- `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.
- `args`  
  *graphics-state* parameters passed as keyword arguments.

**Description**  
The `copy-pixels` function copies a rectangular area from one port to another. The `transform`, `mask`, `mask-transform`, `compositing-mode` and `shape-mode` from `to-port`'s `graphics-state` are used.

The `(to-x to-y)` is transformed according to `to-port`'s transform. When `to-port`'s `drawing-mode` is `:quality` the target is generally fully transformed, except that when it copies from a visible window it may generate expose events when copying from an obscured part, and in `drawing-mode` `:quality` it ignores the transformation in this case.

If `to-port`'s `drawing-mode` is `:compatible` then the image is not scaled or rotated. For more information about `drawing-mode`, see "Drawing mode and anti-aliasing" in the CAPI User Guide.
The to-port and from-port need not be the same depth and can be the same object. The from-x and from-y values are interpreted as pixel positions in the window coordinates of from-port, that is, they are not transformed by from-port’s transform.

Notes  

**copy-pixels** can be used to draw to an **output-pane** inside the display-callback of that pane, but it cannot be used to copy from the **output-pane** inside its display-callback (the result of such an operation is not defined).

See also  

copy-area  
output-pane

---

**copy-transform**  

Function (inline)

Summary  

Returns a copy of a transform.

Package  

**graphics-ports**

Signature  

copy-transform transform => result

Arguments  

transform  

A transform.

Values  

result  

A transform.

Description  

The **copy-transform** function returns a copy of transform.

---

**create-pixmap-port**  

Function

Summary  

Creates a pixmap port and its window system representation.

Package  

**graphics-ports**
Signature: `create-pixmap-port pane width height &key background collect relative clear drawing-mode => pixmap-port`

Arguments:
- `pane`: A graphics port for a window.
- `width`: An integer.
- `height`: An integer.
- `background`: A color designator.
- `collect`: A boolean.
- `relative`: A boolean.
- `clear`: A list or `t`.
- `drawing-mode`: One of the keywords `:compatible` and `:quality`.

Values: `pixmap-port` A pixmap graphics port.

Description:
The `create-pixmap-port` function creates a pixmap-port and its window system representation. The `pane` argument specifies the color-user, used for color conversions, and its representation may also be used by the library to match the pixmap port properties. The value of `background` is used to initialize the `graphics-state-background`.

If `clear` is `t`, the pixmap is cleared to its background color, otherwise the initial pixel values will be non-deterministic. If `clear` is a list of the form `(x y width height)`, only that part of the pixmap is cleared initially. The default value is `nil`.

If `relative` is non-`nil`, the pixmap graphics port collects pixel coordinates corresponding to the left, top, right, and bottom extremes of the drawing operations taking place within the body forms, and if these extend beyond the edges of the pixmap (into negative coordinates for example) the entire drawing is offset by an amount which ensures it remains within the port. It is as if the port moves its relative origin in
order to accommodate the drawing. If the drawing size is greater than the screen size, then some of it is lost. The default value is \texttt{nil}.

If \texttt{collect} is non-\texttt{nil}, this causes the drawing extremes to be collected but without having the pixmap shift to accommodate the drawing, as \texttt{relative} does. The extreme values can be read using the \texttt{get-bounds} function, and \texttt{make-image-from-port}.

See also \texttt{with-pixmap-graphics-port}

\*\texttt{default-image-translation-table}*

\textbf{Variable}

\textbf{Summary} \ The default image translation table.

\textbf{Package} \ graphics-ports

\textbf{Description} \ The \texttt{*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 \texttt{load-image}

\textbf{define-font-alias}

\textbf{Function}

\textbf{Summary} \ Defines an alias for a font.

\textbf{Package} \ graphics-ports

\textbf{Signature} \ define-font-alias \ keyword \ font

\textbf{Arguments} \ \begin{itemize}
\item \texttt{keyword} \ A keyword.
\item \texttt{font} \ A font.
\end{itemize}
2  GP Reference Entries

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.

**Notes**

Dithers do not affect drawing or the anti-aliasing 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.
- `args` `graphics-state` parameters passed as keyword arguments.
- `filled` A boolean.

---

697
2  GP Reference Entries

Description
The draw-arc function draws an arc contained in the rectangle from \((x \, y)\) to \((x+\text{width} \, y+\text{height})\) 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. When port’s drawing-mode is :quality the arc is transformed properly, but if drawing-mode is :compatible and 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, mask, shape-mode and compositing-mode from the port’s graphics-state are all used, unless overridden in args. Additionally on Unix only, stipple and pattern are used. When filled is non-nil, a sector is drawn.

See also
draw-arcs
make-graphics-state

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 graphics-state parameters passed as keyword arguments.
The `draw-arcs` function draws several arcs as specified by the `description` sequence. This is usually more efficient than making several calls to `draw-arc`. The `description` argument is a sequence of values of the form `x y width height start-angle sweep-angle`. See `draw-arc` for more information.

**draw-character**

**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` `graphics-state` parameters passed as keyword arguments.

**Description**

The `draw-character` function draws the character `character` at `(x y)` on the port. The `transform, foreground, background, operation, stipple, pattern, mask, mask-transform, font, text-mode` and `compositing-mode` from the port’s `graphics-state` are all used, unless overridden in `args`.

`(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.
The `graphics-state` parameter `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`

**Arguments**

- `port` A graphics port.
- `x` A real number.
- `y` A real number.
- `radius` A real number.
- `args` `graphics-state` parameters passed as keyword arguments.
- `filled` A boolean.

**Description**

The `draw-circle` function draws a circle of the given radius centered on `(x y)`. The `transform`, `foreground`, `background`, `operation`, `thickess`, `scale-thickness`, `mask`, `shape-mode` and `compositing-mode` from the port's `graphics-state` are all used, unless overridden in `args`. When `filled` is non-nil, the circle is filled with the foreground color.

**Notes**

`draw-circle` does not work properly under a rotation transform (see `make-transform`). A workaround is to use a many-sided polygon drawn by `draw-polygon` which will be rotated correctly.

**Example**

`(gp:draw-circle port 100 100 20)`
(gp:draw-circle port 100 100 50
   :filled t
   :foreground :green)

See also  graphics-state

draw-ellipse

Function

Summary
Draws an ellipse.

Package  graphics-ports

Signature
\texttt{draw-ellipse port x y x-radius y-radius \&rest args \&key filled}

Arguments
- \texttt{port}  A graphics port.
- \texttt{x}  A real number.
- \texttt{y}  A real number.
- \texttt{x-radius}  A real number.
- \texttt{y-radius}  A real number.
- \texttt{radius}  A real number.
- \texttt{args}  \texttt{graphics-state} parameters passed as key-word arguments.
- \texttt{filled}  A boolean.

Description
The \texttt{draw-ellipse} function draws an ellipse of the given radii centered on \((x, y)\). The \texttt{transform}, \texttt{foreground}, \texttt{background}, \texttt{operation}, \texttt{thickness}, \texttt{scale-thickness}, \texttt{mask}, \texttt{shape-mode} and \texttt{compositing-mode} from the port's \texttt{graphics-state} are all used, unless overridden in \texttt{args}. When \texttt{filled} is non-\texttt{nil}, the ellipse is filled with the foreground color.
Notes  

**draw-ellipse** does not work properly under a rotation transform when *port’s drawing-mode* is :compatible. A workaround is to use a many-sided polygon drawn by **draw-polygon** which will be rotated correctly.

**draw-ellipse** does work properly under any transform when *port’s drawing-mode* is :quality.

See **make-transform** for information about rotation transforms.

For more information about *drawing-mode*, see "Drawing mode and anti-aliasing" in the CAPI User Guide.

Example

```
(gp:draw-ellipse port 100 100 20 40)
(gp:draw-ellipse port 100 100 50 10
  :filled t
  :foreground :green)
```

See also  

**graphics-state**

**draw-image**  

Function

Summary  

Displays an image on a graphics port at a given position.

Package  

**graphics-ports**

Signature  

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  

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>port</em></td>
<td>A graphics port.</td>
</tr>
<tr>
<td><em>image</em></td>
<td>An image.</td>
</tr>
<tr>
<td><em>to-x</em></td>
<td>A real number.</td>
</tr>
<tr>
<td><em>to-y</em></td>
<td>A real number.</td>
</tr>
<tr>
<td><em>args</em></td>
<td><strong>graphics-state</strong> parameters passed as keyword arguments.</td>
</tr>
</tbody>
</table>
from-x A real number.
from-y A real number.
to-width A real number.
to-height A real number.
from-width A real number.
from-height A real number.
global-alpha A real number in the inclusive range \([0,1]\), or nil.

Description

The **draw-image** function displays *image* on the port at *to-x* to-\(y\).

The default value of *from-x* and *from-y* is 0. The *width* and *height* arguments default to the size of the image.

When *port’s* *drawing-mode* is :compatible, graphics state translation is guaranteed to be supported but support for scaling and rotation are library dependent. Specifically, scaling is supported in the Windows, Cocoa and GTK+ implementations, but not on X11/Motif.

When *port’s* *drawing-mode* is :quality, the target coordinates are fully transformed according to the transformation in the *graphics-state*.

For more information about *drawing-mode*, see “Drawing mode and anti-aliasing” in the **CAPI User Guide**.

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.
Notes

On Microsoft Windows, if the image was loaded from a .ico file then `draw-image` ignores `from-x`, `from-y`, `from-width`, `from-height` and the `graphics-state` operation when drawing the image, and also `global-alpha` is ignored.

draw-line

Function

Summary

Draws a line between two given points.

Package

`graphics-ports`

Signature

`draw-line port from-x from-y to-x to-y &rest args`

Arguments

- `port` | A graphics port.
- `from-x` | A real number.
- `from-y` | A real number.
- `to-x` | A real number.
- `to-y` | A real number.
- `args` | `graphics-state` parameters passed as keyword 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`, `mask`, `shape-mode` and `compositing-mode` are used. Additionally on Unix only, `stipple` and `pattern` are used.

See also

`draw-lines`

`graphics-state`
**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` `graphics-state` parameters passed as keyword 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-path**  
*Function*

Summary: Draws a path at a given point, optionally closing it or filling it.

Package: **graphics-ports**

Signature: `draw-path port path x y &rest args &key closed filled fill-rule`

Arguments:
- `port` A graphics port.
- `path` A path specification.
- `x` A real number.
- `y` A real number.
The function draw-path draws the path path at (x y) in port. When closed is non-nil, a line is drawn from the last point in the path to the start of the last figure in the path. When filled is non-nil, the path is filled, otherwise its outline is drawn; the closed argument is ignored if filled is non-nil. transform, foreground, background, thickness, scale-thickness, dashed, dash, line-end-style, line-joint-style and mask from port’s graphics state (see graphics-state) are all used. fill-rule specifies how overlapping regions are filled. Possible values for fill-rule are :even-odd and :winding.

path is a path specification, which consists of path elements that describe a number of disconnected figures. The origin of the path is (x y), so all other coordinates within the path are translated relative to that point.

The following formats of path specification are supported:

- A sequence of lists, each of which is a path element as described below.

- A function designator to generate the path elements. Graphics ports calls the function when it wants to obtain the path elements. The function takes a single argument, which is a function that should be called with each path elements as its arguments.

The following path elements can be used:

:close Closes the current figure by adding a straight line from the current point to the start point.
:move x y  Closes the current figure and starts a new one at (x y).

:line x y  Adds a straight line to the current figure, from the current point to (x y) and makes (x y) be the current point.

:arc x y width height start-angle sweep &optional movep
  Adds an elliptical arc to the current figure, contained in the rectangle from (x y) to (x+width y+width) from start-angle to start-angle+sweep-angle. Both angles are specified in radians and positive values mean anti-clockwise. If movep is nil (the default), then a straight line is also added from the current point to the start of the arc, otherwise a new figure is started from the start of the arc. The end of the arc becomes the new current point.

:bezier cx1 cy1 cx2 cy2 x y
  Adds a cubic Bézier curve to the current figure, from the current point to (x y) using control points (cx1 cy1) and (cx2 cy2).

:rectangle x y width height
  Adds a self contained figure, a rectangle from (x y) to (x+width y+width).

:ellipse x y x-radius y-radius
  Adds a self contained figure, an ellipse of the given radii centered on (x y).

:scale sx sy elements
  Adds the path elements elements, scaling them by sx and sy.

:rotate theta elements
  Adds the path elements elements, rotating them theta radians about the origin. If theta is positive, then the rotation is clockwise.
\textbf{translate} \textit{dx dy elements}

Adds the path elements \textit{elements}, translating them by \textit{dx} and \textit{dy}.

\textbf{transform} \textit{transform elements}

Adds the path elements \textit{elements}, transformed by \textit{transform}.

\textbf{Examples}

Draws two lines from (40 30) to (140 30) and from (140 30) to (140 130):

\begin{verbatim}
(draw-path port '((:line 100 0) (:line 100 100)) 40 30)
\end{verbatim}

Draws an outline triangle with vertices (40 30), (140 30) and (140 130):

\begin{verbatim}
(draw-path port '((:line 100 0) (:line 100 100))
               40 30 :closed t)
\end{verbatim}

Draws a filled triangle with vertices (40 30), (140 30) and (140 130):

\begin{verbatim}
(draw-path port '((:line 100 0) (:line 100 100))
               40 30 :filled t)
\end{verbatim}

Draws a filled triangle exactly as in the previous example but using a function to generate the path elements:

\begin{verbatim}
(flet ((generate (fn)
          (funcall fn :line 100 0)
            (funcall fn :line 100 100)))
    (draw-path port #'generate 40 30 :filled t))
\end{verbatim}

Draws 6 copies of a shape consisting of two lines and an arc:

\begin{verbatim}
(labels ((generate-1 (fn)
            (funcall fn :line 50 0)
            (funcall fn :line 50 50)
            (funcall fn :arc 0 -50 100 100
                       (/ pi -2) (/ pi -2)))
       (generate-6 (fn)
                    (dotimes (x 6)
                      (funcall fn :rotate (* 2 pi (/ x 6))
                                   #'generate-1))))
    (draw-path port #'generate-6 80 80))
\end{verbatim}
There are more examples in

examples/capi/graphics/paths.lisp

See also
draw-polygon
draw-line
draw-arc
draw-ellipse
graphics-state

draw-point

Function

Summary
Draws a pixel or unit square at a given point.

Package
graphics-ports

Signature
draw-point port x y &rest args

Arguments
| port       | A graphics port. |
| x          | A real number.   |
| y          | A real number.   |
| args       | graphics-state parameters passed as keyword arguments.

Description
The draw-point function draws a single-pixel point at (x y). The transform, foreground, background, operation, mask, shape-mode and compositing-mode graphics-state parameters are used. Additionally on Unix only, stipple and pattern are used.

When drawing-mode is :compatible the output is a single pixel. Note that its position is transformed in the normal way.

When drawing-mode is :quality this draws a unit square as if by draw-rectangle, transformed in the normal way.

See also
draw-points
graphics-state
**draw-points**

*Function*

**Summary**

Draws pixels or unit squares at given points.

**Package**

`graphics-ports`

**Signature**

`draw-points port description &rest args`

**Arguments**

- `port` A graphics port.
- `description` A description sequence.
- `args` `graphics-state` parameters passed as keyword arguments.

**Description**

The `draw-points` function draws several points (as if by `draw-point`) as specified by the `description` argument, which is a sequence of `x y` pairs. It is usually faster than several calls to `draw-point`. See `draw-point` for more information.

**See also**

`draw-point`

---

**draw-polygon**

*Function*

**Summary**

Draws a polygon.

**Package**

`graphics-ports`

**Signature**

`draw-polygon port points &rest args &key filled closed fill-rule`

**Arguments**

- `port` A graphics port.
- `points` A description sequence.
- `filled` A boolean.
- `closed` A boolean.
- `fill-rule` A keyword.
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`, `mask`, `shape-mode` and `compositing-mode` from port’s `graphics-state` are all used, unless overridden in `args`. Additionally on Unix only, `stipple` and `pattern` are used.

`fill-rule` specifies how overlapping regions are filled. Possible values are `:even-odd` and `:winding`.

See also

- `draw-polygons`
- `graphics-state`

**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.
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, mask, shape-mode and compositing-mode from the port’s graphics-state are all used, unless overridden in args. Additionally on Unix only, stipple and pattern are used.

fill-rule specifies how overlapping regions are filled. Possible values are :even-odd and :winding.

Example

This draws two hexagons, one inside the other:

```
(gp:draw-polygons oo
   '((150 100 200 100 235 150 200 200 150 200 115 150)
     (140 90 210 90 250 150 210 210 140 210 100 150))
  :closed t)
```

See also

draw-polygon
draw-rectangle

Function

**draw-rectangle**

Summary

Draws a rectangle.

Package

graphics-ports
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**: graphics-state parameters passed as keyword arguments.

Description

The **draw-rectangle** function draws a rectangle whose corners are \((x, y), (x+width, y), (x+width, y+height)\) and \((x, y+height)\).

If **filled**, 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-join-style**, **mask**, **shape-mode** and **compositing-mode** are used. Additionally on Unix only, **stipple** and **pattern** are used.

See also

- **draw-rectangles**
- **graphics-state**

**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**: `graphics-state` parameters passed as keyword 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`.

`filled`, 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`, `mask`, `shape-mode` and `compositing-mode` are used. Additionally on Unix only, `stipple` and `pattern` 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.
Description
Draws the string with the baseline starting at \((x \ y)\). The `transform`, `foreground`, `background`, `operation`, `stipple`, `pattern`, `mask`, `mask-transform`, `font`, `text-mode` and `compositing-mode` from `port`'s `graphics-state` are all used, unless overridden in `args`.

`start` and `end` specify which elements of the `string` to draw. The default value of `start` is 0.

`block`, if non-nil, causes each character to be drawn in a character cell filled with the `background` of `port`'s `graphics-state`.

You can draw with the system highlight by setting `graphics-state` parameter `foreground :color_highlight`text and `background :color_highlight`.

Notes
The `graphics-state` parameter `operation` is not supported for drawing text on Microsoft Windows.

Example

```lisp
(let ((op (capi:contain
          (make-instance 'capi:output-pane
                         :background :red)))))
(gp:draw-string op "highlighted"
10 10
:graphics-args
(list :foreground
      :color_highlighttext))
```

See also `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
      `ensure-gdiplus`, and
   b) `force` is `nil`, and
   c) `event-func` was either not passed or is `eq` to the value
      that was passed for point a)

      then `ensure-gdiplus` simply returns `nil`.

3. If GDI+ was already started, shut it down.

4. Start GDI+, and return the result of `GdiplusStartup`.
   This is 0 for success. For the meaning of other values, see
   the documentation of `gpStatus` in the MSDN.

   If `shutdown` is true, then if GDI+ was started `ensure-gdiplus
   shuts it down, and returns `t`, otherwise `ensure-gdiplus
   returns `nil`. The default value of `shutdown` is `nil`.

   The default value of both `event-func` and `force` is `nil`.

   See also `read-external-image`

---

**external-image**

**Class**

**Summary**

A class representing a color image.

**Package**

`graphics-ports`

**Description**

The class `external-image` provides a representation of a
color image that is subject to `write-external-image`, `read-
external-image` and `convert-external-image` operations.

**See also**

`convert-external-image`

`read-external-image`

`write-external-image`
**external-image-color-table**

*Function*

**Summary**
Returns a vector containing RGB color specifications of an external image.

**Package**
`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 replace- 
ment-color-table must be the same length as the external image’s original color table. It is a vector of RGB color-specifi- 
cations.

externalize-and-write-image  
Function

Summary  
Externalizes and writes an image to file.

Package  
graphics-ports

Signature  
externalize-and-write-image gp image filename &key type if-
exists errorp x-hot y-hot quality &allow-other-keys => result

Arguments  

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gp</td>
<td>A CAPI pane.</td>
</tr>
<tr>
<td>image</td>
<td>An image object.</td>
</tr>
<tr>
<td>filename</td>
<td>A file namestring or a pathname.</td>
</tr>
<tr>
<td>type</td>
<td>One of the keywords :bmp, :jpg, :jpeg, :png and :tiff. Other keywords may be supported, depending on the platform.</td>
</tr>
<tr>
<td>errorp</td>
<td>A boolean.</td>
</tr>
<tr>
<td>x-hot</td>
<td>A non-negative integer.</td>
</tr>
<tr>
<td>y-hot</td>
<td>A non-negative integer.</td>
</tr>
<tr>
<td>quality</td>
<td>An integer in the range [0,100].</td>
</tr>
</tbody>
</table>

Values  
result A filename or nil.
The function `externalize-and-write-image` externalizes and writes an `image` object to file.

The output image type can be specified by the argument `type`. If `type` is not supplied then the output image type is determined by the file type in the `filename`.

If `type` is supplied, it must be a keyword which specifies a known type, as returned by `list-known-image-formats` with `for-writing-too` t. The types :bmp, :jpg, :png and :tiff are known on all platforms (except Motif). Additionally, :jpeg is an as alias for :jpg.

If `type` is not supplied, then the file extension of the `filename` is used to "guess" the type. In general it is the extension uppercased and interned in the keyword package. It also recognizes some special cases:

<table>
<thead>
<tr>
<th>File extension</th>
<th>Image type</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;TIF&quot;</td>
<td>:tiff</td>
</tr>
<tr>
<td>&quot;DIB&quot;</td>
<td>:bmp</td>
</tr>
<tr>
<td>&quot;JPE&quot;</td>
<td>:jpg</td>
</tr>
<tr>
<td>&quot;JPEG&quot;</td>
<td>:jpg</td>
</tr>
<tr>
<td>&quot;JFIF&quot;</td>
<td>:jpg</td>
</tr>
<tr>
<td>&quot;JP2&quot;</td>
<td>:jpg2000</td>
</tr>
</tbody>
</table>

Table 2.1 Image type from file extension: special cases

**Note:** Image type :jpg2000 is implemented on Cocoa only. Errorp controls what happens if `externalize-and-write-image` does not recognize the type. If `errorp` is non-nil, it calls `error`, otherwise it returns `nil`. The default value of `errorp` is t.
if-exists controls what to do if the filename already exists, in the same way as the if-exists argument to open. However, unlike open, the default value of if-exists is :supersede.

x-hot and y-hot are used only when generating a CUR file, which is currently implemented on GTK+ only. They specify the hotspot coordinates when the image is used as a cursor (in a LispWorks application by load-cursor and (setf capi:simple-pane-cursor), or in other applications). Their values must be integers within the width/height of the image. The default value of both x-hot and y-hot is 0.

quality is used for writing a JPG image on GTK+. It must be an integer in the inclusive range [0,100]. High values generate better images and larger files.

result is filename on success, or nil for an unknown type when errorp is nil. It signals an error in other cases (for example, failure to open the file because of permissions).

Examples
There is a simple example in:
examples/capi/graphics/images-with-alpha.lisp

See also
list-known-image-formats

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`

---

**f2pi**

**Constant**

**Summary**

\[ (* 2 \, pi) \text{ as a single-float} \]

**Package**

`graphics-ports`

**Description**

The constant `f2pi` is the result of \( (\text{float} (* 2.0 \, \text{cl:pi}) 1.0) \). It is a `cl:single-float`. 

---

722
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 function find-best-font 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.

By default find-best-font looks only for Truetype fonts in LispWorks 6.1 and later.

Notes
With the default drawing-mode :quality only Truetype fonts are supported. Non-Truetype fonts are supported only when using drawing-mode :compatible.

Compatibility note
To get the LispWorks 6.0 behavior where non-Truetype fonts are also found, pass :type :wild to make-font-description.

Examples
examples/capi/graphics/catherine-wheel.lisp

See also
fpi
fpi-by-2
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

Type

Summary An object corresponding to a font in the native system.

Description font objects are returned by find-best-font and find-matching-fonts.
font objects are used to specify fonts for drawing, either in the graphics-state of the port or in the drawing functions themselves. Font objects can also be used for querying the actual attributes of the font (ascent, descent, etc) and the dimensions of character and strings.

Notes

Font objects are not externalizable objects.

See also

- font-description
- find-best-font
- find-matching-fonts
- graphics-state
- get-font-ascent
- get-font-descent
- get-font-width
- get-font-height
- get-font-average-width
- get-char-width
- get-char-ascent
- get-char-descent
- get-character-extent
- get-string-extent
- compute-char-extents
- font-single-width-p
- font-fixed-width-p
- font-description

**font-description**

**Type**

**Summary**

An object used in CAPI to describe a font.

**Description**

Objects of type **font-description** contain a description of a font. The description can be partial, with only some attributes given values. **font-description** objects are the normal way of specifying fonts in CAPI.
font-description objects are created or returned by make-font-description, convert-to-font-description, font-description, merge-font-descriptions and augment-font-description.

font-description objects are used as the font specification for CAPI panes (see simple-pane). They can also be used directly in calls to find-best-font and find-matching-fonts.

Notes
1. font-description objects do not contain native system dependent values, and are externalizable objects.
2. A font-description cannot be used directly as an argument to draw-string or draw-character, or as the value of the graphics state parameter font in a graphics-state. These require the result of find-best-font or find-matching-fonts.

See also
make-font-description
convert-to-font-description
font-description
merge-font-descriptions
augment-font-description
font-description-attributes

font-description

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.
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  

```language
font-description-attributes fdesc => font-attributes
```

Arguments  

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>fdesc</code></td>
<td>A font description.</td>
</tr>
</tbody>
</table>

Values  

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>font-attributes</code></td>
<td>A list of font attributes.</td>
</tr>
</tbody>
</table>

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  

```language
font-description-attribute-value fdesc font-attribute => value
```

Arguments  

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>fdesc</code></td>
<td>A font description.</td>
</tr>
</tbody>
</table>


font-attribute A font attribute.

Values value A font attribute value.

Description The font-description-attribute-value function returns the value of font-attribute in fdesc, or :wild if font-attribute is not specified in fdesc.

---

font-dual-width-p

Function

Summary The predicate for dual-width fonts.

Signature font-dual-width-p port &optional font => result

Arguments

- port A graphics port.
- font A font or a font-description object.

Values result A boolean.

Description The function font-dual-width-p returns t when the font is fixed-width and contains double width characters. Such a font is dual-width.

See also font-fixed-width-p
font-single-width-p

---

font-fixed-width-p

Function

Summary The predicate for fixed-width fonts.

Package graphics-ports

Signature font-fixed-width-p port &optional font => result
Arguments

- `port` A graphics port.
- `font` A `font` or a `font-description` object.

Values

- `result` A boolean.

Description

The function `font-fixed-width-p` returns `t` if the optionally specified `font`, or otherwise the font associated with `port`, is fixed-width.

Fixed-width is not exactly the same as single-width. A fixed-width font with double width characters is dual-width; other fixed-width fonts are single-width.

Notes

`editor-pane` supports variable width fonts on Microsoft Windows, GTK+ and Motif.

See also

- `font-description`
- `font-single-width-p`

---

**font-single-width-p**

*Function*

Summary

The predicate for single-width fonts.

Signature

`font-single-width-p port &optional font => result`

Arguments

- `font` A `font` or a `font-description` object.

Values

- `result` A boolean.

Description

The function `font-single-width-p` returns `t` when all characters in the font specified by `font` are of the same width.

A single-width font is fixed-width.

See also

- `font-fixed-width-p`
- `font-description`
fpi  

**Constant**

Summary  
pi as a single-float.

Package  
graphics-ports

Description  
The constant fpi is the result of (float cl:pi 1.0). It is a cl:single-float.

See also  
2pi  
f2pi  
fpi-by-2

fpi-by-2  

**Constant**

Summary  
(/ pi 2) as a single-float

Package  
graphics-ports

Description  
The constant fpi-by-2 is the result of (float (* 0.5 cl:pi) 1.0). It is a cl:single-float

See also  
fpi  
f2pi

free-image  

**Function**

Summary  
Frees the library resources allocated with an image.

Package  
graphics-ports

Signature  
free-image port image

Arguments  
port  
A CAPI pane.
image  
An image.
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**

```scheme
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**

```scheme
get-bounds pixmap-port => left, top, right, bottom
```

**Arguments**

- `pixmap-port`  
  A graphics port.

**Values**

- `left`  
  An integer.
An integer.

right  An integer.

top    An integer.

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

<table>
<thead>
<tr>
<th>Summary</th>
<th>Returns the extent of a character in pixels.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Package</td>
<td><code>graphics-ports</code></td>
</tr>
<tr>
<td>Signature</td>
<td><code>get-character-extent</code> <code>port</code> <code>character</code> &amp;optional <code>font</code> =&gt; <code>left</code>, <code>top</code>, <code>right</code>, <code>bottom</code></td>
</tr>
<tr>
<td>Arguments</td>
<td><code>port</code> A CAPI pane.</td>
</tr>
<tr>
<td></td>
<td><code>character</code> A character.</td>
</tr>
<tr>
<td></td>
<td><code>font</code> A font.</td>
</tr>
</tbody>
</table>
Values  

<table>
<thead>
<tr>
<th>Value</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-character-extent** function returns the extent in pixels of the *character* in the font associated with *port*, or the *font* given.

---

**get-char-ascent**  

**Function**

**Summary**  
Returns the ascent of a character in pixels.

**Package**  
*graphics-ports*

**Signature**  

```
get-char-ascent port character font => ascent
```

**Arguments**  

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>port</td>
<td>A CAPI pane.</td>
</tr>
<tr>
<td>character</td>
<td>A character.</td>
</tr>
<tr>
<td>font</td>
<td>A font.</td>
</tr>
</tbody>
</table>

**Values**  

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ascent</td>
<td>An integer.</td>
</tr>
</tbody>
</table>

**Description**  
The **get-character-ascent** function returns the ascent in pixels of the *character* in the font associated with *port*, or the *font* given.

---

**get-char-descent**  

**Function**

**Summary**  
Returns the descent of a character in pixels.

**Package**  
*graphics-ports*
Signature        get-char-descent  port  character  font  =>  descent

Arguments        port               A CAPI pane.
                 character           A character.
                 font                A font.

Values           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

Function

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.
### get-enclosing-rectangle

**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.

**Description**  
The `get-font-ascent` function returns the ascent in pixels of the font associated with `port`, or the `font` given.
get-font-average-width  
Function

Summary  Returns the average width of a font in pixels.

Package  graphics-ports

Signature  get-font-average-width port &optional font => average-width

Arguments  

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>port</td>
<td>A CAPI pane.</td>
</tr>
<tr>
<td>font</td>
<td>A font.</td>
</tr>
</tbody>
</table>

Values  

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>average-width</td>
<td>An integer.</td>
</tr>
</tbody>
</table>

Description  The get-font-average-width function returns the average-width in pixels of the font associated with port, or the font given.

get-font-descent  
Function

Summary  Returns the descent in pixels of a font.

Package  graphics-ports

Signature  get-font-descent port &optional font => descent

Arguments  

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>port</td>
<td>A CAPI pane.</td>
</tr>
<tr>
<td>font</td>
<td>A font.</td>
</tr>
</tbody>
</table>

Values  

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>descent</td>
<td>An integer.</td>
</tr>
</tbody>
</table>

Description  The get-font-descent function returns the descent in pixels of the font associated with port, or the font given.
get-font-height

Function

Summary
Returns the height of a font.

Package
graphics-ports

Signature
get-font-height port &optional font => height

Arguments
port A CAPI pane.
font A font.

Values
height An integer.

Description
The get-font-height function returns the height in pixels of the font associated with port, or the font given.

get-font-width

Function

Summary
Returns the width of a font.

Package
graphics-ports

Signature
get-font-width port &optional font => width

Arguments
port A graphics port.
font A font.

Values
width An integer.

Description
The function get-font-width 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. `get-graphics-state` is deprecated. Use `port-graphics-state` instead.

**Package**  
`graphics-ports`

**Signature**  
`get-graphics-state port => state`

**Arguments**  
`port`  
A graphics port.

**Values**  
`state`  
A `graphics-state` object.

**Description**  
`get-graphics-state` is deprecated. Use `port-graphics-state` instead.

**See also**  
`port-graphics-state`

get-origin  

**Summary**  
Returns the coordinate origin of a pixmap graphics port.

**Package**  
`graphics-ports`

**Signature**  
`get-origin pixmap-port => x y`

**Arguments**  
`pixmap-port`  
A graphics port.

**Values**  
`x`  
An integer.

`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 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

```
(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*

<table>
<thead>
<tr>
<th>Summary</th>
<th>Returns the overall scaling factor of a transform.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Package</td>
<td><code>graphics-ports</code></td>
</tr>
<tr>
<td>Signature</td>
<td><code>get-transform-scale</code> transform =&gt; result</td>
</tr>
<tr>
<td>Arguments</td>
<td><code>transform</code> A <code>transform</code> object.</td>
</tr>
<tr>
<td>Values</td>
<td><code>result</code> A real number.</td>
</tr>
<tr>
<td>Description</td>
<td>The <code>get-transform-scale</code> function returns a single number representing the overall scaling factor present in the <code>transform</code>.</td>
</tr>
</tbody>
</table>

**graphics-port-background**  
**graphics-port-font**  
**graphics-port-foreground**  
**graphics-port-transform**  
*Functions*

<table>
<thead>
<tr>
<th>Summary</th>
<th>Accesses the <code>background</code>, <code>font</code>, <code>foreground</code> or <code>transform</code> in the graphics state of a graphics port.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Package</td>
<td><code>graphics-ports</code></td>
</tr>
</tbody>
</table>
| Signature | `graphics-port-background` port => color-spec  
`graphics-port-font` port => font                                                                 |

740
graphics-port-foreground  port  =>  color-spec
graphics-port-transform  port  =>  transform
(setf graphics-port-background)  color-spec  port  =>  color-spec
(setf graphics-port-font)  font  port  =>  font
(setf graphics-port-foreground)  color-spec  port  =>  color-spec
(setf graphics-port-transform)  transform  port  =>  transform

Arguments

- **port**
  A graphics port.

Values

- **color-spec**
  A color specification, or *nil*.
- **font**
  A *font* object, or *nil*.
- **transform**
  A *transform* object.

Description

The functions *graphics-port-background*, *graphics-port-font*, *graphics-port-foreground* and *graphics-port-transform* access the background, font, foreground or transform in the *graphics-state* associated with *port*. This can be used to set the value by *setf*.

See the *graphics-state* entry for the types and acceptable values of the specific slots.

See also

- **graphics-state**
- **port-graphics-state**
- **set-graphics-state**
- **transform**
- **with-graphics-state**

---

**graphics-state**

Function

**Summary**

The graphics state object, holding default parameters for drawing operations on an associated *port*.

**Package**

*graphics-ports*
Slots

transform

A transform object which determines the coordinate transformation applying to the graphics port. The default value is the unit transform which leaves the port coordinates unchanged from those used by the host window system — origin at top left, X increasing to the right and Y increasing down the screen. Allowed values are anything returned by the transform functions, described in section "Graphics state transforms" of the CAPI User Guide.

foreground

Determines the foreground color used in drawing functions. The value can be a pixel value, a color name symbol, a color name string or a color spec object. Using pixel values results in better performance. The default value is :black. The value :color_highlighttext is useful for drawing text with the system highlighting.

background

Determines the background color used in drawing functions which use a stipple. Valid values are the same as for foreground. The default value is :white. The value :color_highlight is useful for drawing text with the system highlighting.

operation

Determines the color combination used in the drawing primitives when the port's drawing-mode is :compatible. 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 CAPI User Guide shows how to use operation.
**stipple**

A 1-bit pixmap ("bitmap") or **nil** (which is the default value). The bitmap is used in conjunction with the **fill-style** when drawing. Here, **nil** means that all pixels are drawn in the **foreground** color. A stipple is not transformed by the **transform** parameter. Its origin is assumed to coincide with the origin of the port. The **stipple** is tiled across the drawing. **stipple** is ignored if a **pattern** is given. If no **fill-style** is given, or it is specified as **solid**, when a **stipple** is given, then **fill-style** defaults to **opaque-stippled**.

**fill-style**

Determines how the drawing is done. The value should be one of **solid**, **opaque-stippled**, **stippled** or **tiled**. The default value **solid** means that the **foreground** is used everywhere. **opaque-stippled** means that the **stipple** bitmap is used with stipple 1s giving the **foreground** and 0s the **background**. **stippled** means that the **stipple** bitmap is used with **foreground** where there are 1s and where 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 **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-joint-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
nil, or a list specifying a shape inside which the drawing is done.
**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.

The `mask-x` parameter works only when the `drawing-mode` is `compatible` and the platform is GTK+ or X11/Motif.

`mask-x` is deprecated.

**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.

The `mask-y` parameter works only when the `drawing-mode` is `compatible` and the platform is GTK+ or X11/Motif.

`mask-y` is deprecated.

**mask-transform**
A `transform` object which determines the coordinate transformation use for the mask in `drawing-mode` `quality`.

**font**
Either `nil` or a `font` object to be used by the `draw-character` and `draw-string` functions. The default value is `nil`.

Note that `font` cannot be a `font-description`. Use `find-best-font` to convert a `font-description` to a `font`.

**text-mode**
A keyword controlling the mode of rendering text, most importantly anti-aliasing.

**shape-mode**
A keyword controlling the mode of drawing shapes (that is, anything except text).

**compositing-mode**
A keyword controlling the combining of new drawing with existing drawing.
Each graphics port has a `graphics-state` object associated with it, providing the default values of graphics parameters for drawing operations. The drawing operations such as `draw-ellipse`, `draw-rectangle` and `draw-string` can override specific parameters by passing them as keyword arguments.

`graphics-state` objects are used in the `with-graphics-state` macro and modified using the accessor functions listed above. See "Setting the graphics state" in the CAPI User Guide for examples.

`mask` should be `nil` (the default), a list of the form `(x y width height)`, defining a rectangle inside which the drawing is done or a list of the form `(:path path :fill-rule fill-rule)` specifying a path inside which the drawing is done. The mask is not tiled.
In the latter case path should be a path specification (see draw-path). The fill-rule specifies how overlapping regions are filled. Possible values are :even-odd and :winding. The mask will be transformed by the mask-transform parameter.

There some examples of path masks in examples/capi/graphics/paths.lisp.

mask-transform is used only in drawing-mode :quality. It is ignored in drawing-mode :compatible. The default value is the unit transform, which can also be specified as nil. Other allowed values include anything returned by the transform functions, described in the section "Graphics state transforms" of the CAPI User Guide. The other allowed value of mask-transform is the keyword :dynamic which is replaced by the current value of the transform graphics state parameter when the drawing operation uses the mask.

Each of text-mode and shape-mode can be one of:

:plain No anti-aliasing.
:antialias With anti-aliasing.
:fastest Fastest rendering. The same as :plain except on Windows
:best Best display.
:default The system default (which is :antialias).

Additionally text-mode can be :compatible, which causes text to be drawn the way it would be drawn if drawing-mode was :compatible. This makes a difference only on Microsoft Windows, because on other platforms the default text-mode draws like the :compatible one.

The default of both text-mode and shape-mode is :default.

compositing-mode is a keyword or an integer controlling the compositing mode, that is the way that a new drawing is combined with the existing value in the target of the drawing to generate the result.
Two values of *compositing-mode* are supported on all platforms other than Motif:

- **:over** Draw over the existing values. If the source is a solid color, then the result is simply the source. If the source has alpha value *alpha*, then it is blended with the destination, with the destination multiplied by the remainder of the alpha, that is (- 1 *alpha*).

- **:copy** The source is written to the destination ignoring the existing values. If the source has alpha and the target does not, that has the effect of converting semi-transparent source to solid.

The default value of *compositing-mode* is **:over**.

The **:copy** value of *compositing-mode* is especially useful for creating a transparent or semi-transparent * pixmap-port*, which can be displayed directly or converted to an image by * make-image-from-port*.

On Cocoa 10.5 and later and GTK+ 2.8 or later, these additional keyword values of *compositing-mode* are supported:

- **:clear**, **:over**, **:in**, **:out**, **:atop**, **:dest-over**, **:dest-in**, **:dest-out**, **:dest-atop**, **:xor** and **:add**. These correspond to the *CAIRO_OPERATOR_* operators in Cairo, which are documented in cairographics.org/operators and the *CGBlendMode* values which are documented in the CGContext Reference at developer.apple.com.

**Note:** on GTK+, the "unbounded" operators (:in, :out, :dest-in and :dest-atop) do not work properly for shape drawings. They can only be used for image drawing and copying operations.

Both Cocoa and GTK+ also allow *compositing-mode* to be an integer, which is simply passed through to the underlying system. This allows using modes that are not available via keywords, but it is not portable. For Cocoa, it is a *CGBlend-
Mode as documented in the CGContext Reference. For GTK+ it is cairo_operator_t, as documented in the entry for cairo_t in the Gnome documentation for Cairo.

Note: For drawing images on Cocoa, only values that corresponding to available keywords work properly.

Notes

1. operation is not supported for drawing text on Microsoft Windows.
2. stipple and fill-style are supported only on X11/Motif.
3. mask-x and mask-y are supported only on GTK+ and X11/Motif, and only when the drawing-mode is :compatible.
4. pattern is supported only on Microsoft Windows, GTK+ and X11/Motif.
5. operation is not supported by Cocoa/Core Graphics so this slot or argument is ignored on Cocoa.
6. operation is ignored when the port’s drawing-mode is :quality.
7. text-mode and shape-mode are supported only on Cocoa, Cairo and GDI+, which are used on Macintosh, GTK and Windows respectively when the drawing-mode is :quality. For more information about drawing-mode, see "Drawing mode and anti-aliasing" in the CAPI User Guide.

Examples

examples/capi/graphics/compositing-mode-simple.lisp
examples/capi/graphics/compositing-mode.lisp

See also

make-graphics-state
set-graphics-state
with-graphics-state
image

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.

Notes
On Cocoa and GTK+ you can drag and drop images. See set-drop-object-supported-formats for more information.

See also
convert-external-image
draw-image
load-image
make-image-from-port
make-sub-image
read-and-convert-external-image

image-access-height

image-access-width

Functions

Summary
Return the dimensions of the underlying image in an Image Access object.

Package
graphics-ports

Signature
image-access-height image-access ==> height
**image-access-width**

```lisp
image-access-width  image-access => width
```

**Arguments**

- `image-access`  An Image Access object

**Values**

- `height`  An integer.
- `width`  An integer.

**Description**

The functions `image-access-height` and `image-access-width` return the height and width of the underlying image in `image-access`.

`image-access` must be an Image Access object returned by `make-image-access`.

**Notes**

It is an error to call `image-access-height` or `image-access-width` on an Image Access object that has been freed by `free-image-access`.

**Example**

See these example files:
- `examples/capi/graphics/image-access.lisp`
- `examples/capi/graphics/image-access-alpha.lisp`

**See also**

- `free-image-access`
- `make-image-access`

---

**image-access-pixel**

**Function**

**Summary**

Gets and sets the pixels in an Image Access object.

**Package**

`graphics-ports`

**Signature**

- `image-access-pixel image-access x y => color-rep`
- `(setf image-access-pixel) color-rep image-access x y => color-rep`

**Arguments**

- `image-access`  An Image Access object
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`
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 \((\times 4 width height)\) where **width** and **height** represent the size of **image-access**.

**Note:** **image-access-pixels-to-bgra** must be called after this function (similarly to (**setf image-access-pixel**)). **image-access** must be an Image Access object returned by **make-image-access**.

**Example**

See the file

```
examples/capi/graphics/image-access-bgra.lisp
```

**See also**

**image-access-pixel**  
**image-access-pixels-to-bgra**

---

**image-access-pixels-to-bgra**

**Function**

**Summary**

Copies pixel values from an Image Access object into a vector.

**Package**

**graphics-ports**

**Signature**

```
image-access-pixels-to-bgra image-access vector
```
Arguments

image-access An Image Access object.
vector A vector.

Description

The function `image-access-pixels-to-bgra` copies all the pixels in the Image Access object `image-access` into the vector `vector` as a sequence of integer values in the range 0-255 for the blue, green, red and alpha components of each pixel. This function is optimized for the case where `vector` has element type `(unsigned-byte 8)`.

An error is signalled if `vector` is not of the correct length for the Image Access object, that is `(* 4 width height)` where `width` and `height` represent the size of `image-access`.

Note: `image-access-pixels-from-bgra` must be called before this function (similarly to `image-access-pixel`). `image-access` must be an Image Access object returned by `make-image-access`.

Example

See the file `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 image-access`

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`.

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`
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**

*Function*

**Summary**

Initialize dither objects up to a given order.

**Package**

`graphics-ports`

**Signature**

`initialize-dithers &optional order`

**Arguments**

`order` An integer.

**Description**

The `initialize-dithers` function initializes dither objects up to the given `order` (size = 2 ^ `order`). By default, order is 3.

**Note:** dithers do not affect drawing or the anti-aliasing that occurs when drawing in Cocoa.

See also `dither-color-spec` `make-dither` `with-dither`

**inset-rectangle**

*Function (inline)*

**Summary**

Moves the corners of a rectangle inwards by a given amount.

**Package**

`graphics-ports`

**Signature**

`inset-rectangle rectangle dx dy &optional dx-right dy-bottom`
Arguments

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 top right 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  

**Summary**  
Invalidates the rectangle associated with the object, which causes it to be redisplayed.

**Package**  
`graphics-ports`

**Signature**  
`invalidate-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**  
By default, the generic function `invalidate-rectangle` invalidates the whole rectangle, but this can be limited by passing the optional arguments.

The effect of invalidating an area is to cause the area to be redrawn. It has no effect on `pixmap-port`. When the pane has a supplied `display-callback`, this callback is called with an area containing the area specified by the argument to `invalidate-rectangle`. However, the call to `display-callback` is asynchronous, and the system coalesces areas from calls to `invalidate-rectangle` and actual expose events, so there is not a one-to-one relation between calls to `invalidate-rectangle` and invocations of `display-callback`.

In general, `invalidate-rectangle` should not be called inside the `display-callback`. If it is called, it must be conditional, otherwise this will cause repeated redisplay.
Notes With `drawing-mode :quality`, drawings are done with antialiasing, which means that they affect pixels which are not obviously part of the drawing. For example, drawing a rectangle with $x = 10$ may affect the pixel at $x = 9$. This needs to be taken into account when computing the arguments to `invalidate-rectangle`.

For pinboard objects the recommended way of forcing redraw is `redraw-pinboard-object`, which takes antialiasing into account.

See also `validate-rectangle`

---

**invert-transform**  
*Function*

**Summary** Constructs the inverse of a transform.

**Package** `graphics-ports`

**Signature** `invert-transform transform &optional into => inverse`

**Arguments**

- `transform`: A `transform` object.
- `into`: A `transform` object or `nil`.

**Values** `inverse`: A `transform` object.

**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

list-known-image-formats Function

Summary Returns the known image formats.

Package graphics-ports

Signature list-known-image-formats screen-spec &optional for-writing-too => formats

Arguments screen-spec A CAPI object, a plist, or nil.
for-writing-too A generalized boolean.

Values formats A list of keywords.

Description The function list-known-image-formats returns a list of keywords which specify known image formats.
screen-spec is an object that convert-to-screen can recognize, typically a pane or simply nil.

If for-writing-too is not supplied or is nil, then formats is a list of formats that can be loaded. All the formats in the list can be loaded, but on Cocoa and Windows the list is not exhaustive, and it may be possible to load formats that are not listed.

If for-writing-too is supplied as non-nil, then formats is a list of types that externalize-and-write-image can write. In this case the list is exhaustive on all platforms, and externalize-and-write-image can write a format if and only if it appears in the list.

All platforms (except Motif) can read and write :bmp, :jpg, :png and :tiff images, and also recognize :jpeg as an alias for :jpg, so the list will always include all of these keywords.

See also convert-to-screen externalize-and-write-image

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

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>port</td>
<td>A graphics port or CAPI object.</td>
</tr>
<tr>
<td>id</td>
<td>A keyword, string or pathname.</td>
</tr>
<tr>
<td>width</td>
<td>The desired width in pixels, or nil.</td>
</tr>
<tr>
<td>height</td>
<td>The desired height in pixels, or nil.</td>
</tr>
</tbody>
</table>

Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>image</td>
<td>An image object.</td>
</tr>
</tbody>
</table>
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.

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.

When `id` specifies a file and `width` and `height` are specified, then the most appropriate image is chosen from the icon file and is scaled accordingly. If `width` and `height` are `nil` the first image in the file is used at its natural size. `width` defaults to `nil` and `height` defaults to `width`.

**Note:** `load-icon-image` is defined only in LispWorks for Windows.

See also

- `draw-image`
- `free-image`
- `load-image`
load-image  

Function

Summary  
Loads an image and returns the image object.

Package  
graphics-ports

Signature  
load-image gp id &key cache type editable image-translation-table => image

Arguments  

- **gp**  
  A graphics port.

- **id**  
  An image identifier, a file, an external-image, or an image.

- **cache**  
  A boolean.

- **type**  
  A keyword, or nil.

- **editable**  
  One of the keywords :with-alpha and :without-alpha, or a boolean.

- **image-translation-table**  
  An image translation table.

Values  

- **image**  
  An image object.

Description  
The load-image function loads an image identified by id via the image-translation-table using the image load function registered with it. It returns an image object with the representation slot initialized. The gp argument specifies a graphics port used to identify the library. It also specifies the resource in which colors are defined and if necessary allocated for the image. If id is in the table but the translation is not an external image, and the image loader returns an external image as the second value, that external image replaces the translation in the table. The default value of image-translation-table is *default-image-translation-table*. 
id can be an image, which is just associated with the port gp and returned if it is a Plain Image or if editable is nil. Otherwise a new Plain Image object is returned, as described below.

id can also be a string or pathname denoting a file, and in this case the image is loaded according to type, as described below.

The cache argument controls whether the image translation is cached. See the convert-external-image function for more details.

type tells load-image that the image is in a particular graphics format. Currently the only recognised value is :bmp, which means the image is a Bitmap. Other values of type cause load-image to load the image according to the file type of id, if id denotes a file, as described for read-external-image. See the Graphics Ports chapter in the CAPI User Guide for a discussion of image handling. The default value of type is nil.

editable controls whether the image image is a Plain Image suitable for use with the Image Access API. The values of editable have the following effects:

nil The image is not editable.

:without-alpha The image is editable, but does not have an alpha channel.

:with-alpha The image is editable and has an alpha channel. It will be fully opaque when loading files without an alpha channel.

Given an image my-image, call
(load-image port my-image :editable t)

to create an image guaranteed to work with make-image-access. The default value of editable is nil.

Normally the image is freed automatically, when gp is destroyed. However there are circumstances where you need to explicitly free an image, for example when you want it to go away before the port. If the image is not freed, a memory leak occurs.

Note: gp must already be created at the time load-image is called. If you need to delay loading the image, for example if you are computing the image dynamically, then you can call load-image in the create-callback of the port or even in its first display-callback.

Compatibility note In LispWorks 4.4 there is a keyword argument :force-plain with the same effect as :editable. :force-plain is still accepted in LispWorks 6.1 for backwards compatibility, but you should now use :editable instead.

See also convert-external-image
*default-image-translation-table*
load-icon-image
make-image
make-image-access

make-dither Function

Summary Makes a dither matrix of a given size.

Package graphics-ports

Signature make-dither size => matrix

Arguments size An integer.
<table>
<thead>
<tr>
<th>Values</th>
<th>matrix</th>
<th>A dither matrix.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>The <em>make-dither</em> function makes a dither matrix of the given size.</td>
<td></td>
</tr>
</tbody>
</table>

**Note:** dithers do not affect drawing or the anti-aliasing 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**

```lisp
make-font-description &rest font-attribute* => fdesc
```

**Arguments**
- `font-attribute` A font attribute.

**Values**
- `fdesc` A font description object.

**Description**
The *make-font-description* function returns a new font description object containing the given font attributes. There is no error checking of the attributes at this point.

The :stock attribute is handled specially: it is omitted from *fdesc*, unless it is the only attribute specified.

**See also**
- augment-font-description
- convert-to-font-description
- find-best-font
find-matching-fonts
font-description
merge-font-descriptions

make-graphics-state

Function

Summary
Creates a graphics-state object.

Package
graphics-ports

Signature
make-graphics-state &key transform foreground background operation thickness scale-thickness dashed dash line-end-style line-joint-style mask fill-style stipple pattern mask-x mask-y font text-mode shape-mode compositing-mode mask-transform => state

Arguments
See graphics-state for interpretation of the arguments.

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).

See also
graphics-state
set-graphics-state

make-image

Function

Summary
Makes a new, empty, image object.

Package
graphics-ports
Signature  \texttt{make-image \textit{port} width height &\texttt{key\ alpha} => image} \\
Arguments  \textit{port} \hspace{1cm} \text{A graphics port.} \\
\textit{width} \hspace{1cm} \text{A positive integer.} \\
\textit{height} \hspace{1cm} \text{A positive integer.} \\
\textit{alpha} \hspace{1cm} \text{A generalized boolean.} \\
Values  \textit{image} \hspace{1cm} \text{An \texttt{image} object.} \\
Description  \texttt{make-image} makes a new blank, editable \texttt{image} object associated with \textit{port} and of the given \textit{width} and \textit{height}. On Windows and Cocoa, if \textit{alpha} is true, then the image will have an alpha channel. The initial pixels in \textit{image} are undefined. \textit{image} is editable, that is, it is suitable for use with the Image Access API. To set the pixels, see \texttt{make-image-access}. \\
See also  \texttt{load-image} \\
\texttt{make-image-access} \\
\textbf{make-image-access} \hspace{1cm} \textit{Generic Function} \\
Summary  \text{Creates an Image Access object.} \\
Package  \texttt{graphics-ports} \\
Signature  \texttt{make-image-access \textit{port} \textit{image} => image-access} \\
Arguments  \textit{port} \hspace{1cm} \text{A graphics port.} \\
\textit{image} \hspace{1cm} \text{An \texttt{image} object.} \\
Values  \textit{image-access} \hspace{1cm} \text{An Image Access object.}
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 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
image-access-height
image-access-pixel
load-image
make-image

**make-image-from-port**

*Function*

**Summary**
Makes an image out of a specified rectangle of a graphics port’s contents.

**Package**
graphics-ports
Signature

\texttt{make-image-from-port port \&optional x y width height => image}

Arguments

\texttt{port} A graphics port.
\texttt{x} An integer.
\texttt{y} An integer.
\texttt{width} An integer.
\texttt{height} An integer.

Values

\texttt{image} An image.

Description

The \texttt{make-image-from-port} function makes an \texttt{image} out of the specified rectangle of the port’s contents. The default is the whole port, but a region can be specified using \texttt{x}, \texttt{y}, \texttt{width}, and \texttt{height}. The default value of \texttt{x} and \texttt{y} is 0.

Normally the image is freed automatically, when \texttt{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 \texttt{externalize-image}

\textbf{make-sub-image} \hspace{1cm} \textit{Function}

Summary

Makes a new image from part of an image.

Package \texttt{graphics-ports}

Signature

\texttt{make-sub-image port image \&optional x y width height => sub-image}

Arguments

\texttt{port} A graphics port.
\texttt{image} An \texttt{image}. 
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**

\[
\text{make-transform} \; \&\text{optional} \; a \; b \; c \; d \; e \; f \Rightarrow \text{transform}
\]

**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 object. |

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 \((x \ y \ 1)\).

Example

This transform will cause rotation by \(\pi/4\) radians:

```lisp
(let ((s (sin (/ pi 4)))
      (c (cos (/ pi 4))))
  (gp:make-transform c s (- s) c 0 0))
```

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.
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

pi-by-2

Constant

Summary  (/ pi 2) as a double-float.

Package  graphics-ports

Description  The constant pi-by-2 is the result of (/ cl:pi 2). It is a cl:double-float.
See also

2pi
fpi

pixblt

Function

Summary
Copies one area of a graphics port to another area of a different graphics port.

pixblt is deprecated.

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.

operation is ignored when the drawing-mode is :quality (the default). See the "Graphics state" section in the CAPI User Guide for valid values for operation.
Pixblt is deprecated, because the :quality drawing-mode does not support operation, and because it ignores the transformations, which means it does not always work as expected. In particular, it can draw at the wrong place inside the display-callback of output-pane.

Use instead copy-area, which does take account of the transform. See also graphics-state parameter compositing-mode for a way to control how copy-area blends the source and the target.

See also copy-area
graphics-state

**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-drawing-mode-quality-p**  
Generic Function

Summary Tests whether a port does quality drawing.

Package graphics-ports

Signature port-drawing-mode-quality-p port => result
### port-drawing-mode-quality-p

**Arguments**

- `port` A graphics port.

**Value**

- `result` A boolean.

**Description**

The generic function `port-drawing-mode-quality-p` returns true if the graphics port `port` does quality drawing.

A port does quality drawing if both

1. it was not made with `drawing-mode :compatible`, and
2. the underlying library supports quality drawing.

Microsoft Windows and Cocoa always support quality drawing, GTK+ supports it from version 2.8 and greater, but Motif never supports it.

**Examples**

```lisp
(exampyle-file "capi/graphics/images-with-alpha")
```

**See also**

Section "Drawing mode and anti-aliasing" in the CAPI User Guide.

---

### port-graphics-state

**Function**

**Summary**

Returns the `graphics-state` object for a graphics port.

**Package**

`graphics-ports`

**Signature**

`port-graphics-state port => state`

**Arguments**

- `port` A graphics port.

**Values**

- `state` A `graphics-state` object.

**Description**

The function `port-graphics-state` returns the `graphics-state` object for `port`. The individual slots can be accessed using the accessor functions documented for `graphics-state`. 
See also \texttt{graphics-state}

\textbf{port-height} \hspace{1cm} \textit{Function}

\textbf{Summary} \hspace{0.5cm} Returns the pixel height of a port.

\textbf{Package} \hspace{0.5cm} \texttt{graphics-ports}

\textbf{Signature} \hspace{0.5cm} \texttt{port-height port \Rightarrow result}

\textbf{Arguments} \hspace{0.5cm} \textit{port} \hspace{1cm} A graphics port.

\textbf{Values} \hspace{0.5cm} \textit{result} \hspace{1cm} An integer.

\textbf{Description} \hspace{0.5cm} The function \texttt{port-height} returns the pixel height of \textit{port}.

\textbf{port-string-height} \hspace{1cm} \textit{Function}

\textbf{Summary} \hspace{0.5cm} Returns the height of a string drawn to a given port in pixels.

\textbf{Package} \hspace{0.5cm} \texttt{graphics-ports}

\textbf{Signature} \hspace{0.5cm} \texttt{port-string-height port string \Rightarrow height}

\textbf{Arguments} \hspace{0.5cm} \textit{port} \hspace{1cm} A graphics port.
\hspace{1cm} \textit{string} \hspace{1cm} A string.

\textbf{Values} \hspace{0.5cm} \textit{height} \hspace{1cm} An integer.

\textbf{Description} \hspace{0.5cm} The \texttt{port-string-height} function returns the \textit{height} in pixels of \textit{string} when drawn to \textit{port}. The font used is the \textit{font} currently in the port’s \texttt{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 font currently in the port’s `graphics-state`.

**Notes**  
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 function `port-width` 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` object.
- transform2: A `transform` object.

**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`.

\[
\text{transform1} = \text{transform1} \cdot \text{transform2}
\]

**premultiply-transforms** *Function*

**Summary**
Premultiplies two transforms.

**Package**
graphics-ports

**Signature**
premultiply-transforms transform1 transform2

**Arguments**
- transform1: A `transform` object.
- transform2: A `transform` object.

**Description**
The `premultiply-transforms` function premultiplies the partial 3 x 3 matrix represented by `transform1` by the partial 3 x 3 matrix represented by `transform2`, storing the result in
transform1. In the result, the translation, scaling and rotation operations contained in transform2 are effectively performed before those in transform1.

\[ \text{transform1 = transform2 . transform1} \]

**read-and-convert-external-image**

**Function**

**Summary**

Returns an image converted from an external image read from a file.

**Package**

`graphics-ports`

**Signature**

`read-and-convert-external-image gp file &key transparent-color-index => image, external-image`

**Arguments**

- `gp` A CAPI pane.
- `file` A pathname designator.
- `transparent-color-index` An integer or `nil`.

**Values**

- `image` An image.
- `external-image` An external image.

**Description**

Returns an image converted from an external image read from `file`. The external image is returned as a second value. `transparent-color-index` is interpreted as described for `read-external-image`.

**See also**

- `convert-external-image`
- `external-image`
- `read-external-image`
Function

**read-external-image**

<table>
<thead>
<tr>
<th>Summary</th>
<th>Returns an external image read from a file.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Package</td>
<td><strong>graphics-ports</strong></td>
</tr>
<tr>
<td>Signature</td>
<td><strong>read-external-image</strong> <em>file</em> &amp;key <strong>transparent-color-index</strong> <strong>type</strong> =&gt; <strong>image</strong></td>
</tr>
<tr>
<td>Arguments</td>
<td><strong>file</strong> A pathname designator.</td>
</tr>
<tr>
<td></td>
<td><strong>transparent-color-index</strong> An integer or nil.</td>
</tr>
<tr>
<td></td>
<td><strong>type</strong> A keyword, or nil.</td>
</tr>
<tr>
<td>Values</td>
<td><strong>image</strong> An external image.</td>
</tr>
<tr>
<td>Description</td>
<td>The <strong>read-external-image</strong> function returns an external image read from <em>file</em>. <strong>transparent-color-index</strong> specifies the index of the transparent color in the color map. <strong>transparent-color-index</strong> 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. <strong>type</strong> tells <strong>read-external-image</strong> 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 <strong>type</strong> cause <strong>read-external-image</strong> to read the image according to the file type of <em>file</em>. &quot;bmp&quot; or &quot;dib&quot; 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 CAPI User Guide for details. The default value of <strong>type</strong> is nil.</td>
</tr>
<tr>
<td>Example</td>
<td>To see the effect of <strong>transparent-color-index</strong>, edit <strong>examples/capi/graphics/images.lisp</strong>.</td>
</tr>
</tbody>
</table>
Specify a non-white :background for the viewer pane. Use an image editing tool to find the transparent color index (183 in this image) and change the call to read-external-image like this:

(gp:read-external-image file :transparent-color-index 183)

Then compile and run the example, click the Change... button and select the Setup.bmp file.

See also  external-image

rectangle-bind  

Macro

Summary  Binds four variables to the corners of a rectangle across a body of code.

Package  graphics-ports

Signature  rectangle-bind ((a b c d) rectangle) &body body => result

Arguments  
v

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>A variable.</td>
</tr>
<tr>
<td>b</td>
<td>A variable.</td>
</tr>
<tr>
<td>c</td>
<td>A variable.</td>
</tr>
<tr>
<td>d</td>
<td>A variable.</td>
</tr>
<tr>
<td>rectangles</td>
<td>A rectangle.</td>
</tr>
<tr>
<td>body</td>
<td>A body of code.</td>
</tr>
</tbody>
</table>

Values  result  The return value of the last form in 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
(setq rectangle-top) top rectangle => top

Arguments
rectangle A rectangle.

top A real number.

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.
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 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  

**Macro**

**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  

**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.
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
deviantRandom

Arguments
port A graphics port.
left A real number.
top A real number.
right A real number.
bottom A real number.

Description
The generic function set-graphics-port-coordinates modifies the transform of the graphics port port permanently such that the edges of port correspond to the values of the other arguments.

Example
The following code

(set-graphics-port-coordinates port :left -1.0
:top 1.0
:right 1.0
:bottom -1.0)
changes the coordinates of the port so that the point (0 0) is in the exact center of the port and the edges are a unit distance away, with a right-handed coordinate system.

By default, \textit{left} and \textit{top} are 1.

\textbf{set-graphics-state} \hspace{1cm} \textit{Function}

\begin{description}
\item[Summary] Directly alters the \texttt{graphics-state} of a graphics port according to the keyword arguments supplied.
\item[Package] \texttt{graphics-ports}
\item[Signature] \texttt{set-graphics-state port \&rest args \&key transform foreground background operation stipple pattern fill-style thickness scale-thickness dashed dash line-end-style line-joint-style mask mask-x mask-y font shape-mode text-mode compositing-mode mask-transform}
\item[Arguments] \texttt{port} \hspace{0.5cm} A graphics port.
\item[Description] The function \texttt{set-graphics-state} directly alters the graphics state of \texttt{port} according to the values of the keyword arguments \texttt{args}. Unspecified keywords leave the associated slots unchanged.

See \texttt{graphics-state} for valid values for \texttt{args}.

See also \texttt{graphics-state} \newline \texttt{with-graphics-state}
\end{description}

\textbf{transform} \hspace{1cm} \textit{Type}

\begin{description}
\item[Summary] The transform type, defined for transform objects.
\item[Package] \texttt{graphics-ports}
\end{description}
The transform type is the type defined for transform objects, which are six-element lists of numbers.

See also graphics-port-transform

**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 **transform-distance** function transforms the distance `(dx dy)` by the rotation and scale in the *transform*. The translation in the transform is ignored. Transformed `(dx dy)` is returned as two values.

### transform-distances

**Function**

**Summary**

Transforms a list of alternating distance vectors by a given transform.

**Package**

`graphics-ports`

**Signature**

`transform-distances transform distances => result`

**Arguments**

- **transform**: A transform.
- **distances**: A list of pairs of real numbers.

**Values**

- **result**: A list of pairs of real numbers.

**Description**

The **transform-distances** function transforms a list of alternating `(dx dy)` pairs in *distances* by the *transform*. Transformed values are returned as a new list.

### transform-is-rotated

**Function**

**Summary**

Returns `t` if a given transform contains a rotation.
Package | graphics-ports
---|---
Signature | transform-is-rotated transform => bool
Arguments | transform A transform.
Values | 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 | graphics-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.
### transform-points

**Signature**

\[
\text{transform-points\ transform\ points\ &optional\ into\ =>\ result}
\]

**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 `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.

**Function**

**Signature**

\[
\text{transform-rect\ transform\ left\ top\ right\ bottom\ =>\ left2\ top2\ right2\ bottom2}
\]

**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.

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.
A real number.

rectangle A rectangle.

The union-rectangle macro modifies the rectangle to be the union of rectangle and \((\text{left top right bottom})\).

\*unit-transform* Variable

Summary The list \((1\ 0\ 0\ 1\ 0\ 0)\).

Package graphics-ports

Signature \*unit-transform*

Description The \*unit-transform* variable holds the list \((1\ 0\ 0\ 1\ 0\ 0)\) which is the unit transform \(I\), such that \(X = XI\), where \(X\) is a 3-vector. Graphics ports are initialized with the unit transform in their graphics-state. This means that port coordinate axes are initially the same as the window axes.

unit-transform-p Function

Summary Returns \(t\) if a given transform is a unit transform.

Package graphics-ports

Signature unit-transform-p transform => bool

Arguments transform A transform.

Values bool A boolean.

Description The unit-transform-p returns \(t\) if transform is the unit transform.
unless-empty-rect-bind

**Macro**

**Summary**
Binds the elements of a rectangle to four variables, and if the rectangle has a non-zero area, executes a body of code.

**Package**
graphics-ports

**Signature**
unless-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.
\[dx\] A real number.
\[dy\] A real number.

Values
\[x\] A real number.
\[y\] A real number.

Description
The \textbf{untransform-distance} function transforms the distance \((dx\ dy)\) by the rotation and scale of the effective inverse of \textit{transform}. The translation in the inverse transform is ignored. The transformed distance \((dx\ dy)\) is returned as two values.

\textbf{untransform-distances}

\textit{Function}

Summary
Transforms a list of integer pairs representing distances by the inverse of a transform.

Package \texttt{graphics-ports}

Signature
\texttt{untransform-distances transform distances => result}

Arguments
\textit{transform} A transform.

\textit{distances} A list of pairs of real numbers.

Values
\textit{result} A list of pairs of real numbers.

Description
The \textbf{untransform-distances} function transforms a list of alternating \((dx\ dy)\) pairs in \textit{distances} by the effective inverse of \textit{transform}. Transformed values are returned as a new list.

\textbf{untransform-point}

\textit{Function}

Summary
Transforms a point by multiplying it by the inverse of a given transform.
untransform-point

Function

Transforms a list of points by the inverse of a given transform.

Arguments
- `transform`: A transform.
- `points`: A list of pairs of real numbers.
- `into`: An optional 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

Generic 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).

Notes
validate-rectangle 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 anti-aliasing 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 port’s graphics state across the execution of a body of code.
Package graphics-ports
Signature

\[
\text{with-graphics-mask} \ (\text{port mask} \ \&\text{key mask-x mask-y mask-transform}) \ \&\text{body body} \Rightarrow \text{result}
\]

Arguments

- **port**: A graphics port.
- **mask**: `nil` or a list specifying a shape.
- **mask-x**: An integer. This argument is deprecated.
- **mask-y**: An integer. This argument is deprecated.
- **mask-transform**: `nil`, `t`, the keyword :dynamic, or a transform.
- **body**: A body of Lisp code.

Values

- **result**: The return value of the last form executed in `body`.

Description

The macro `with-graphics-mask` binds the `mask` slot of `port`'s `graphics-state` while evaluating `body`. The mask can be a rectangular area specified by a list of the form \((x y \text{width height})\) or a path specified by a list of the form \((:path path :fill-rule fill-rule)\).

`mask-x` and `mask-y` are deprecated. They work only when the `drawing-mode` is :compatible and the platform is GTK+ or X11/Motif. By default, `mask-x` and `mask-y` are both 0.

The `mask-transform` argument is used to set the `mask-transform` graphics state parameter. If `mask-transform` is `nil`, then the `mask` is not transformed. If `mask-transform` is `t`, then the `mask` is transformed by the current graphics state transform at the time that `with-graphics-mask` is used. If `mask-transform` is :dynamic, then the `mask` is transformed by the graphics state transform that is in effect when the drawing operation uses the mask. Otherwise `mask-transform` should be a transform object. The default value of the `mask-transform` argument is `nil`.

Examples

This example file demonstrates the use of `mask-transform:`
See also graphics-state

with-graphics-post-translation

Macro

Summary
Like with-graphics-translation except that the translation is done after applying all existing transforms.

Signature
with-graphics-post-translation (port dx dy) &body body => result

Arguments
port A graphics port.
dx A real number.
dy A real number.
body Lisp forms.

Values
result The value returned by the last form of body.

Description
The macro with-graphics-post-translation is the same as with-graphics-translation, but the translation is done after applying all existing transforms. That means that the translation is "absolute", not transformed. In contrast, when using with-graphics-translation the translation is transformed by any existing transform(s).

Examples
This form draws a 40x40 rectangle at (100,100), because the scale is applied to the coordinates of the rectangle, but not to the translation.

(gp:with-graphics-scale (port 2 2)
 (gp:with-graphics-post-translation (port 100 100)
  (gp:draw-rectangle port 0 0 20 20))))

Compare with this form, using with-graphics-translation instead, which draws a 40x40 rectangle at (200,200), because the scale applies to the translation too:
with-graphics-rotation

Macro

Summary
Performs a call to apply-rotation with a given angle for the duration of the macro’s body.

Package
graphics-ports

Signature
with-graphics-rotation (port angle) &body body => result

Arguments
- port: A graphics port.
- angle: 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-rotation macro performs a call to
(apply-rotation transform angle)
on the port’s transform for the duration of the body of the macro.

angle is in radians. If angle is positive, then the rotation is clockwise.

Examples
examples/capi/graphics/catherine-wheel.lisp

See also
apply-rotation
with-graphics-scale

Macro

Summary
Performs a call to apply-scale with a given scale for the duration of the macro’s body.

Package
graphics-ports

Signature
with-graphics-scale (port sx sy) &body body => result

Arguments
port A graphics port.
sx A real number.
sy 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-scale macro performs a call to
(apply-scale transform sx sy)
on the port’s transform for the duration of the body of the macro.

See also
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

**with-graphics-state** (port &rest 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 shape-mode text-mode compositing-mode mask-transform) body => result

Arguments

- **port**: A graphics port.
- **body**: A body of Lisp code.

Values

- **result**: The return value of the last form executed in body.

Description

The **with-graphics-state** macro binds the graphics state values for the specified port to the values specified in the **args** list. The keyword arguments **args** correspond to the slots in the graphics state, as described in **graphics-state**.

For example:

```
(with-graphics-state (port :thickness 12 :foreground my-color) ...) 
```

Arguments that are not supplied default to the current state of that slot in the **graphics-state**. The arguments **fill-style** and **stipple** are used only on Unix.

**mask-x** and **mask-y** are deprecated. They work only when the **drawing-mode** is **:compatible** and the platform is GTK+ or X11/Motif.

An extra keyword argument **:state** can be used. The value must be a **graphics-state** object created by a call to **make-graphics-state**. The contents of the **graphics-state** object passed are used instead of the **port's** state.

Example

```
(setf gstate (make-graphics-state))

(setf (graphics-state-foreground gstate) my-color)

(with-graphics-state (port :state gstate)
  (draw-rectangle port image-1 100 100))
```
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 transform associated with the graphics port port with transform during the body of the macro. The port is given a new transform obtained by pre-multiplying its current transform with transform. This has the effect of preceding any translation, scaling and rotation operations specified in the body of the macro by those operations embodied in transform.

See also graphics-state
set-graphics-state
with-graphics-translation
with-graphics-post-translation
with-graphics-scale
with-graphics-rotation
with-graphics-transform
with-graphics-transform-reset
with-graphics-mask
with-graphics-transform-reset

Macro

Summary
Like with-graphics-transform except that it ignores existing transforms.

Signature
with-graphics-transform-reset (port &optional transform)
&body body => result

Arguments
port A graphics port.
transform A transform.
body Lisp forms.

Values
result The value returned by the last form of body.

Description
The macro with-graphics-transform-reset works the same as with-graphics-transform except that it ignores existing transforms.

If the argument transform is nil, the body is applied without transform (that is, with the unit transform).

Examples
This form ignores the translation, and applies only the explicit transform (which is really just scale), so that the overall effect is to draw a 30x20 rectangle at (0,0).

(gp:with-graphics-translation (port 100 100)
 (gp:with-graphics-transform-reset (port (gp:make-transform 3 0 0 2 0 0 ))
 (gp:draw-rectangle port 0 0 10 10)) )

Compare with using with-graphics-transform, which applies both the translation and the explicit transform, so that the overall effect is to draw a rectangle 30x20 at (100,100).
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.

Examples
examples/capi/graphics/catherine-wheel.lisp
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 graphics port.

body  
A body of Lisp code.

Values  
result  
The return value of the last form executed in body.

Description  
The macro with-inverse-graphics 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 port swapped.

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.
The \texttt{with-relative-drawing} macro evaluates the code in \texttt{body} with the \texttt{relative} and \texttt{collect} internal variables of the pixmap graphics port \texttt{port} set to \texttt{nil} to turn off the port's collecting of drawing bounds and automatic shifting of its origins. Use this macro only within a \texttt{with-pixmap-graphics-port} macro.

\textbf{with-pixmap-graphics-port}

\begin{description}
\item[Summary] Binds a port to a new pixmap graphics port for the duration of the macro's code body.
\item[Package] \texttt{graphics-ports}
\item[Signature] \texttt{with-pixmap-graphics-port (port pane width height \&key background collect relative clear drawing-mode) \&body body} => \texttt{result}
\item[Arguments]
\begin{itemize}
\item \texttt{port} A graphics port.
\item \texttt{pane} An output pane.
\item \texttt{width} An integer.
\item \texttt{height} An integer.
\item \texttt{background} A color keyword.
\item \texttt{collect} A boolean.
\item \texttt{relative} A boolean.
\item \texttt{clear} A list or \texttt{t}.
\item \texttt{drawing-mode} One of the keywords \texttt{compatible} and \texttt{quality}.
\item \texttt{body} A body of Lisp code.
\end{itemize}
\item[Values]
\begin{itemize}
\item \texttt{result} The return value of the last form executed in \texttt{body}.
\end{itemize}
\end{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.

The *background* and *foreground* default to the values in the graphics state of *pane*.

**Example**

In the code below the background in *p2* inherits from *p1*, so it draws two green rectangles.

```
(let ((op (capi:contain
           (make-instance 'capi:output-pane
                          :background :red)))))
(sleep 0.1)
(gp:with-pixmap-graphics-port (p1 op 20 30
                                :background :green
                                :clear t)
   (gp:with-pixmap-graphics-port (p2 p1 20 30 :clear t)
      (gp:copy-pixels op p1 10 10 20 30 0 0)
      (gp:copy-pixels op p2 10 60 20 30 0 0)))
```

---

**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*

**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.
with-transformed-area

Summary
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.
port A graphics port.
x A real number.
y A real number.
body A body of Lisp code.

Values
result The return value of the last form executed in body.
The with-transformed-point 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-points

**Macro**

**Summary**
Binds a list of transformed points in a port to a list across the execution of the macro’s body.

**Package**
graphics-ports

**Signature**
with-transformed-points \((points \ port) &body \ body => result\)

**Arguments**
- \(points\): 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 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 \(port\).

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**
with-transformed-rect \((nx1 \ ny1 \ nx2 \ ny2 \ port \ x1 \ y1 \ x2 \ y2) \&body \ body => result\)
Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$nx_1$</td>
<td>A variable.</td>
</tr>
<tr>
<td>$ny_1$</td>
<td>A variable.</td>
</tr>
<tr>
<td>$nx_2$</td>
<td>A variable.</td>
</tr>
<tr>
<td>$ny_2$</td>
<td>A variable.</td>
</tr>
<tr>
<td>$port$</td>
<td>A graphics port.</td>
</tr>
<tr>
<td>$x_1$</td>
<td>A real number.</td>
</tr>
<tr>
<td>$y_1$</td>
<td>A real number.</td>
</tr>
<tr>
<td>$x_2$</td>
<td>A real number.</td>
</tr>
<tr>
<td>$y_2$</td>
<td>A real number.</td>
</tr>
<tr>
<td>$body$</td>
<td>A body of Lisp code.</td>
</tr>
</tbody>
</table>

Values

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$result$</td>
<td>The return value of the last form executed in $body$.</td>
</tr>
</tbody>
</table>

Description

During the evaluation of the `with-transformed-rect` macro $body$, the two points ($x_1, y_1$) and ($x_2, y_2$) are transformed by the port’s current transform and the resulting values bound to the variables named by the $nx_1 ny_1 nx_2 ny_2$ 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

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$external$-image</td>
<td>An external-image.</td>
</tr>
<tr>
<td>$file$</td>
<td>A file.</td>
</tr>
<tr>
<td>$if$-exists</td>
<td>A keyword.</td>
</tr>
</tbody>
</table>
### 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 \texttt{color} package.

\textbf{apropos-color-alias-names}

\textit{Function}

\begin{itemize}
  \item \textbf{Summary} \quad Returns color aliases containing a given string.
  \item \textbf{Package} \quad \texttt{color}
  \item \textbf{Signature} \quad \texttt{apropos-color-alias-names \string \ => \ list}
  \item \textbf{Arguments} \quad \texttt{substring} \quad A string.
  \item \textbf{Values} \quad \texttt{list} \quad A list of symbols.
  \item \textbf{Description} \quad Returns a list of symbols whose symbol-names contain \texttt{substring} and which are defined as aliases in the color-database defining color aliases. By convention these are in the keyword package.
\end{itemize}
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")
 (:ORANGERED3 :ORANGERED1 :INDIANRED3 :INDIANRED1
 :PALEVIOLETRED :RED :INDIANRED :INDIANRED2
 :INDIANRED4 :ORANGERED :MEDIUMVIOLETRED
 :VIOLETRED :ORANGERED2 :ORANGERED4 :RED1 :RED2 :RED3
 :RED4 :PALEVIOLETRED1 :PALEVIOLETRED2 :PALEVIOLETRED3
 :PALEVIOLETRED4 :VIOLETRED3 :VIOLETRED1 :VIOLETRED2
 :VIOLETRED4)

See also
apropos-color-alias-names
apropos-color-spec-names
get-all-color-names

apropos-color-spec-names

Function

Summary
Returns colors containing a given string.

Package
color

Signature
apropos-color-spec-names substring => list

Arguments
substring
A string.

Values
list
A list of symbols.

Description
Returns a list of symbols whose symbol-names contain substring and which are defined as original entries in the color-database defining color aliases. By convention these are in the keyword package.

Example
COLOR 14 > (define-color-alias :mygray100 :gray100)
 (#S(COLOR-ALIAS COLOR :GRAY100))
COLOR 15 > (apropos-color-names "GRAY100")
 (:MYGRAY100 :GRAY100)
COLOR 16 > (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

\texttt{color-red color-spec => color-component}
\texttt{color-green color-spec => color-component}
\texttt{color-blue color-spec => color-component}
\texttt{color-hue color-spec => color-component}
\texttt{color-saturation color-spec => color-component}
\texttt{color-value color-spec => color-component}

Arguments\texttt{ color-spec}  A color specification.

Values\texttt{ color-component}  A color component from the appropriate color model.

Description

If \texttt{color-spec} is not from the appropriate color model (\texttt{:rgb} in the case of \texttt{color-red, color-green} and \texttt{color-blue}, and \texttt{:hsv} in the case of \texttt{color-hue, color-saturation} and \texttt{color-value}) then the component is calculated.

Example

\texttt{COLOR 31 > (color:make-rgb 1.0s0 0.0s0 0.0s0)}
\texttt{#:RGB 1.0S0 0.0S0 0.0S0)}
\texttt{COLOR 32 > (color-red *)}
\texttt{1.0S0}
\texttt{COLOR 33 > (color-green **)}
\texttt{0.0S0}
\texttt{COLOR 34 > (color-value ***)}
\texttt{1.0S0}
\texttt{COLOR 35 > }

See also

\texttt{make-hsv}
\texttt{make-rgb}
\texttt{make-gray}
\texttt{color-model}
\texttt{color-level}

\texttt{*color-database*}

Summary

The current color-database.
Package | color

Description | 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`.

Example | 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*

Summary | 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.

Example | 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 0.66667S0)
COLOR 30 > (color-model *)
:GRAY
COLOR 31 >

See also
make-hsv
make-rgb
make-gray
color-<component>
color-level

**color-with-alpha**

*Function*

Summary
Adds a specified alpha component to a color.

Package
graphics-ports
3  COLOR Reference Entries

Signature  
```lisp
color-with-alpha color alpha => color-spec
```

Arguments  
- `color`: A color specification.
- `alpha`: A real in the inclusive range [0,1].

Values  
- `color-spec`: A color specification, or `nil`.

Description  
The function `color-with-alpha` returns a color like the argument `color` but with alpha component `alpha`.

`color` needs to be a color specification, either a keyword naming a color (a member of the result of calling `get-all-color-names`), or a color-spec (for example the result of `make-rgb`).

`alpha` must be a real in the inclusive range [0,1], otherwise an error is signaled. `alpha = 0` means `color-spec` is transparent, `alpha = 1` means it is solid.

`color-with-alpha` returns a color-spec, or `nil` if `color` is not recognized.

See also  
- `get-all-color-names`
- `make-rgb`

\[\text{colors=}\]

Function  

Summary  
Tests to see if two colors are equal.

Package  
`color`

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

Function

Summary

Return the representation of a color specification on a given graphics port.

Package

color

Signature

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 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.
define-color-alias

Summary
Lets you define an alias for a color specification or alias.

Package
color

Signature
define-color-alias name color &optional if-exists => alias

Arguments
name The name of the new alias.

color A color specification for the new alias.

if-exists This can be one of the following:

:replace — Replace any existing alias.

:error — Raise an error if alias is already defined.

:ignore — Ignore redefinition of an alias.

By default, it is :replace.

Values alias The color alias.

Description Define name to be a color alias for color, which may be another color alias or a color-spec.

Example 1
COLOR 16 > (define-color-alias :mygray :darkslategray)
(#S(COLOR-ALIAS COLOR :DARKSLATEGRAY))
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 :DARKSLATEGREY
   2 Continue, without redefining alias :MYGRAY
   3 Try a new name for the alias, instead of :MYGRAY
   4 (abort) Return to level 0.
   5 Return to top loop level 0.
   6 Destroy process.
Type :c followed by a number to proceed or type :? for other options

COLOR 18 : 1 >

Example 2
COLOR 19 > (define-color-alias :lispworks-blue
(make-rgb 0.70s0 0.90s0 0.99s0))
(#S(COLOR-ALIAS COLOR #(RGB 0.699999S0 0.9S0 0.99S0)))
COLOR 20 >

See also
get-color-alias-translation
get-color-spec

define-color-models
Macro

Summary
Defines 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:

(model-name component-descr*)

A component-descr is a list:
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.

**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 specification for that model is calculated. Thus, `ensure-rgb` returns a color specification in the RGB color model, whatever color model is used in `color-spec`.

If `color-spec` has an alpha component, then `result` has that same alpha component.
ensure-model-color is similar to the above three functions, except that a color-model model is explicitly passed as an argument to the function. The color-spec returned is in the color-model specified by model.

ensure-color returns a color specification for color-spec, in the color model specified by match-color-spec. Thus, color specifications may be converted from one model to another with having to explicitly state the color model.

Example

COLOR 36 > (ensure-hsv (make-rgb 0.70s0 0.90s0 0.99s0))
#:HSV 4.31033S0 0.707069S0 0.99S0)

COLOR 37 > (ensure-gray (make-rgb 0.70s0 0.90s0 0.99s0))
#:GRAY 0.8633318S0)

COLOR 39 > (ensure-model-color (make-rgb 0.70s0 0.90s0 0.99s0) :hsv)
#:HSV 4.31033S0 0.707069S0 0.99S0)

COLOR 43 > (ensure-color (make-hsv 0.70s0 0.90s0 0.99s0) (make-rgb 0.70s0 0.90s0 0.99s0))
#:RGB 0.99S0 0.890999S0 0.92069924)

See also

convert-color

colors=

get-all-color-names

Function

Summary
Returns a list of all color-names in the color database.

Package color

Signature get-all-color-names &optional sort => color-names

Arguments sort If t, sort list of color names alphanumerically. 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 alphanumerically 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 25 > (color:define-color-alias :listener-background :color-background) (#S(COLOR-ALIAS COLOR :COLOR-BACKGROUND))
COLOR 26 > (get-color-alias-translation :listener-background) :LISPWORKS-BLUE
get-color-spec Function

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-rgb 0.70s0 0.90s0 0.99s0))
(#S(COLOR-ALIAS COLOR #:RGB 0.699999S0 0.9S0 0.99S0))
COLOR 29 > (color:define-color-alias :color-background :lispworks-blue)
(#S(COLOR-ALIAS COLOR :LISPWORKS-BLUE))
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)
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 \texttt{make-gray level \&optional alpha => color-spec}

Arguments

- \textit{level}: A color component used to define the gray level required.
- \textit{alpha}: A number between 0 and 1, or \texttt{nil}.

Values \texttt{color-spec}: A color specification.

Description

Return a color-spec in the :GRAY model with component \textit{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.

\textit{alpha} indicates the alpha value of the color. 0 means it is transparent, 1 means it is solid. If \textit{alpha} is \texttt{nil} or not specified then the color does not have an alpha component and it is assumed to be solid.

Example

\texttt{COLOR 25 > (color:make-gray 0.66667S0)}

\#(:GRAY 0.66667S0)

See also \texttt{make-hsv}
\texttt{make-rgb}
\texttt{color-model}
\texttt{color-<component>}
\texttt{color-level}
\texttt{color-alpha}

\textbf{make-hsv}

\textit{Function}

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

```lisp
COLOR 27 > (color:make-hsv 1.2s0 0.5s0 0.9s0)
#:HSV 1.2S0 0.5S0 0.9S0)
```

See also

- `make-rgb`
- `make-gray`
- `color-model`
- `color-<component>`
- `color-level`
- `color-alpha`

### make-rgb

**Function**

Summary

Returns a color specification in the red-green-blue model.

Package

`color`
Signature  
\texttt{make-rgb red green blue \&optional alpha => color-spec}

Arguments  
- \texttt{red}  
A red component.
- \texttt{green}  
A green component.
- \texttt{blue}  
A blue component.
- \texttt{alpha}  
A number between 0 and 1, or \texttt{nil}.

Values  
- \texttt{color-spec}  
A color specification.

Description  
Return a color-spec in the :RGB model with components \texttt{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.

\textit{alpha} indicates the alpha value of the color. 0 means it is transparent, 1 means it is solid. If \textit{alpha} is \texttt{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:

\begin{verbatim}
COLOR 25 > (color:make-rgb 1.0s0 0.0s0 0.0s0)
#(:RGB 1.0S0 0.0S0 0.0S0)
\end{verbatim}

See also  
- \texttt{make-hsv}
- \texttt{make-gray}
- \texttt{color-model}
- \texttt{color-<component>}
- \texttt{color-level}
- \texttt{color-alpha}

\texttt{read-color-db}

Function  

Summary  
Reads the color definitions contained in a file.
read-color-db &optional file => color-database

Arguments
file
A filename or pathname containing the color definitions to be read. If file is not given, read-color-db uses the default color definitions file in the LispWorks library.

Values
color-database
A database definition.

Description
This reads color definitions from the given file (a filename or pathname). The returned data structure can be passed to color:load-color-database. The format of the file is:

```
#(:RGB 1.0s0 0.980391s0 0.980391s0)     snow
#(:RGB 0.972548s0 0.972548s0 1.0s0)     GhostWhite
...
```

Each line contains a color definition which consists of a color-spec and a name. The names are converted to uppercase and interned in the keyword package. Whitespace in names is preserved.

See also
load-color-database
*color-database*
delete-color-translation

unconvert-color

Function

Summary
Returns a color specification for a color representation.

Package
color

Signature
unconvert-color port color-rep => color

Arguments
port
A graphics port.
**color-rep**

A color representation on `port`.

**Values**

- **color**
  
  A color specification.

**Description**

The function `unconvert-color` returns a color specification corresponding to the color representation `color-rep` on the Graphics Port `port`.

If `color-rep` is a color specification, a symbol or a color alias, then it is simply returned since the color system can interpret these directly.

Otherwise `color-rep` is assumed to be a color representation on `port`, like those returned by `convert-color` and `image-access-pixel`, and a corresponding RGB value is returned.

**See also**

- `convert-color`
- `image-access-pixel`
Index

Numerics
2\pi constant 679

A
abort-callback function 1
abort-dialog function 2
abort-exit-confirm function 3
:accelerator initarg 331
Accelerators 251, 331
accepts-focus-p generic function 4
:accepts-focus-p initarg 66, 166
accessor functions
application-interface-application-menu 58
application-interface-dock-menu 58
application-interface-message-callback 58
browser-pane-before-navigate-callback 16
browser-pane-debug 16
browser-pane-document-complete-callback 16
browser-pane-internet-explorer-callback 16
browser-pane-navigate-complete-callback 16
browser-pane-navigate-error-callback 16
browser-pane-new-window-
callback 16
browser-pane-progress-callback 16
browser-pane-status-text-change-callback 16
browser-pane-successful-p 16
browser-pane-title 16
browser-pane-title-change-callback 16
browser-pane-update-commands-callback 16
browser-pane-url 16
button-alternate-callback 468
button-armed-image 27
button-cancel-p 27
button-default-p 27
button-disabled-image 27
button-enabled 27
button-image 27
button-press-callback 468
button-selected 27
button-selected-disabled-image 27
button-selected-image 27
callbacks-action-callback 39
callbacks-callback-type 39
callbacks-extend-callback 39
callbacks-retract-callback 39
callbacks-selection-callback 39
capi-object-name 43
capi-object-plist 43
choice-initial-focus-item 47
choice-interaction 47
choice-selection 47
cocoa-view-pane-init-function 62
cocoa-view-pane-view-class 62
collection-items 66
collection-items-count-function 66
collection-items-get-function 66
collection-items-map-function 66
collection-print-function 66
collection-test-function 66
collector-pane-stream 72
display-pane-text 112
docking-layout-controller 119
docking-layout-divider-p 119
docking-layout-docking-test-function 119
docking-layout-items 119
docking-layout-orientation 119
document-frame-container 124
drawn-pinboard-object-display-callback 136
editor-pane-buffer 159
editor-pane-change-callback 149
editor-pane-composition-face 149
editor-pane-enabled 149
editor-pane-fixed-fill 149
editor-pane-line-wrap-face 149
editor-pane-line-wrap-marker 149
editor-pane-text 149
editor-pane-wrap-style 149
element-interface 167
element-parent 167
element-widget-name 167
filled 174, 478
filtering-layout-matches-text 182
filtering-layout-state 182
form-title-gap 190
form-vertical-adjust 190
form-vertical-gap 190
graph-edge-from 198
graph-edge-to 198
graph-node-height 199
graph-node-in-edges 199
graph-node-out-edges 199
graph-node-width 199
graph-node-x 199
graph-node-y 199
graph-object-element 200
graph-object-object 200
graph-pane-layout-function 201
graph-pane-roots 201
help-key 66, 167, 332, 623
image-height 750
image-pinboard-object-image 219
image-width 750
interactive-pane-stream 224
interactive-pane-top-level-function 224
interface-activate-callback 230
interface-confirm-destroy-function 230
interface-create-callback 230
interface-default-toolbar-states 230
interface-destroy-callback 230
interface-drag-image 230
interface-geometry-change-callback 230
interface-help-callback 230
interface-iconify-callback 230, 240
interface-menu-bar-items 230
interface-message-area 230, 239
interface-override-cursor 230
interface-pathname 230
interface-pointer-documentation-enabled 230
interface-title 230
interface-toolbar-items 230
interface-toolbar-states 230
interface-tooltip-enabled
function 642
tree-view-checkbox-status 642
tree-view-children-function 642
tree-view-expandp-function 642
tree-view-has-root-line 642
tree-view-image-function 642
tree-view-leaf-node-p-function 642
tree-view-retain-expanded-nodes 642
tree-view-right-click-extended-match 642
tree-view-roots 642

:action-callback initarg 38
:action-callback-expand-p initarg 640
:activate-callback initarg 228
activate-pane function 5
:activep initarg 404, 475
active-pane-copy function 6
active-pane-copy-p function 6
active-pane-cut function 6
active-pane-cut-p function 6
active-pane-deselect-all function 6
active-pane-deselect-all-p function 6
active-pane-paste function 6
active-pane-paste-p function 6
active-pane-select-all function 6
active-pane-select-all-p function 6
active-pane-undo function 6
active-pane-undo-p function 6
ActiveX 362
:adjust initarg 73, 498
:adjust item in :buttons initarg 594
:after-input-callback initarg 149
"alive" interface
definition 177
"alive" pane
definition 9
:alternate-callback initarg 467
:alternating-background initarg 273
:alternative initarg 331

:alternative-action-callback initarg 38
analyze-external-image function 680
anti-aliasing 154, 202, 374, 566, 666, 674, 745, 749
text on GTK+ 403
text on Microsoft Windows 403
append-items generic function 7
Application menu 58
application-interface-application-menu accessor function 58
application-interface-dock-menu accessor function 58
application-interface-message-callback accessor function 58
:application-menu initarg 58
apply-in-pane-process function 8
apply-in-pane-process-if-alive function 9
apply-rotation function 680
apply-rotation-around-point function 681
apply-scale function 682
apply-translation function 683
apropos-color-alias-names function 823
apropos-color-names function 824
apropos-color-spec-names function 825
Arguments screen A screen. 574
:armed-image initarg 26
:armed-images initarg 31
arrow-pinboard-object class 10
attach-interface-for-callback function 11
attach-simple-sink function 12
attach-sink function 13
augment-font-description function 683
:auto-arrange-icons initarg 289
:automatic-resize initarg 594, 552
:auto-menus initarg 227
:auto-reset-column-widths initarg 289, 345

B
background graphics state parameter 742
:background initarg 551
beep-pane function 14
:before-change-callback initarg 597
:before-input-callback initarg 149
:before-navigate-callback initarg 15
:best-height initarg 227
:best-width initarg 227
:best-x initarg 227
:best-y initarg 227
:browse-file item in :buttons initarg 592
browser-pane class 15
browser-pane-before-navigate-callback accessor function 16
browser-pane-busy generic function 23
browser-pane-debug accessor function 16
browser-pane-document-complete-callback accessor function 16
browser-pane-go-back generic function 23
browser-pane-go-forward generic function 23
browser-pane-internet-explorer-callback accessor function 16
browser-pane-navigate generic function 23
browser-pane-navigate-complete-callback accessor function 16
browser-pane-navigate-error-callback accessor function 16
browser-pane-new-window-callback accessor function 16
browser-pane-progress-callback accessor function 16
browser-pane-stop generic function 23
browser-pane-successful-p accessor function 16
browser-pane-title accessor function 16
browser-pane-title-change-callback accessor function 16
browser-pane-update-commands-callback accessor function 16
browser-pane-url accessor function 16
:buffer-modes initarg 149
:buffer-name initarg 72, 149
built-in scrolling 196
button class 25
button-alternate-callback accessor function 468
button-armed-image accessor function 27
button-cancel-p accessor function 27
:button-class initarg 31
button-default-p accessor function 27
button-disabled-image accessor function 27
button-enabled accessor function 27
:button-height initarg 620
button-image accessor function 27
button-panel class 31
button-press-callback accessor function 468
:buttons initarg 586
button-selected accessor function 27
button-selected-disabled-image accessor function 27
button-selected-image accessor function 27
:button-width initarg 620
C
calculate-constraints generic function 36
calculate-layout generic function 37
:callback initarg 26, 28, 181, 337, 475, 509, 585, 610, 622
:callback-data-function initarg 337
:callback-object initarg 181
callsbacks 38
for button panels
for buttons
passing different variables
callbacks class
:callbacks initarg 31, 620, 629
callbacks-action-callback accessor function
callbacks-callback-type accessor function
callbacks-extend-callback accessor function
callbacks-retract-callback accessor function
callbacks-selection-callback accessor function
:callback-type initarg 38, 578, 585, 610
call-editor generic function
cancel-button image identifier
:cancel-button initarg 31
:cancel-function initarg in :buttons initarg 593
:cancel-p initarg 26
can-use-metafile-p function
CPI process
capi-object class
capi-object-name accessor function
capi-object-plist accessor function
:caret-position initarg 584
:change-callback initarg 149, 181, 488, 585, 610
:change-callback-type initarg 585
:character-format initarg 488
:checkbox-change-callback initarg 641
:checkbox-child-function initarg 641
:checkbox-initial-status initarg 641
:checkbox-next-map initarg 640
:checkbox-parent-function initarg 640
:checkbox-status initarg 640
check-button class
check-button-panel class
%child% geometry slot
:child initarg 564
:children-function initarg 201, 639
choice class
choice-initial-focus-item accessor function
choice-interaction accessor function
choice-selected-item generic function
choice-selected-item-p function
choice-selected-items generic function
choice-selection accessor function
choice-update-item function
class options
:oclass 101
:definition 93
:interfaces 101
:layouts 93
:menubar 93
:menus 93
:panes 93
:source-interfaces 101
classes
arrow-pinboard-object 10
browser-pane 15
button 25
button-panel 31
callbacks 38
capi-object 42
check-button 44
check-button-panel 45
choice 46
cocoa-default-application-interface 58
cocoa-view-pane 62
collection 65
collector-pane 71
color-screen 73
column-layout 73
display-pane 111
docking-layout 118
document-container 123
document-frame 124
double-headed-arrow-pinboard-object 128
double-list-panel 129
drawn-pinboard-object 136
draw-pane 146
draw-pane 148
element 165
element 174
Cocoa Event Loop process 106
cocoa-default-application-interface class 58
cocoa-view-pane class 62
cocoa-view-pane-init-function accessor function 62
cocoa-view-pane-view function 63
cocoa-view-pane-view-class accessor function 62
collect-interfaces generic function 64
collection class 65
:collection initarg 263
collection-find-next-string generic function 69
collection-find-string generic function 70
collection-items accessor function 66
collection-items-count-function accessor function 66
collection-items-get-function accessor function 66
collection-items-map-function accessor function 66
collection-last-search generic function 70
collection-print-function accessor function 66
collection-search generic function 71
collection-test-function accessor function 66
collector-pane class 71
collector-pane-stream accessor function 72
color-<component> function 826
*color-database* variable 827
:color-function initarg 272
color-level function 826, 828
color-model function 829
colors* function 830
color-screen class 73
color-with-alpha function 829
:column initarg 211
:column-function initarg 345
column-layout class 73
column-layout-divider 74
:columns initarg 345
:combine-child-constraints initarg 575, 578
:command initarg 547
command table 372
complete-button image identifier 595
:complete-do-action initarg 586
:completion item in :buttons initarg 592
:completion-function initarg 584
component-name function 76
:component-name initarg 362, 366
compositing-mode graphics state parameter 745
:composition-callback initarg 373
:composition-face initarg 149
compress-external-image function 687
compute-char-extents function 687
comtab 372
Confirm Before Exiting 77, 519
:confirm-change-function initarg 585
:confirm-destroy-function initarg 227
confirm-pane function 78
confirm-quit function 76
confirm-yes-or-no function 78
constants 2pi 679
f2pi 722
fpi 730
fpi-by-2 730
pi-by-2 776
contain function 79
container 124
container special slot 124
continuation function, dialog creating 662
using 108, 385, 417, 428, 439, 441, 444, 446, 448, 449, 451, 454, 456, 457, 459, 467
:controller initarg 118
convert-color function 831
convert-external-image function 688
convert-relative-position function 80
convert-to-font-description function 689
convert-to-screen function 81
copy-area function 690
copy-external-image function 691
copy-pixels function 692
copy-transform function 693
count-collection-items generic
style* variable 148
editor-pane class 148
editor-pane-blink-rate generic function 158
editor-pane-buffer accessor function 159
editor-pane-change-callback accessor function 149
editor-pane-composition-face accessor function 149
*editor-pane-composition-selected-range-face-plist* variable 160
*editor-pane-default-composition-face* variable 162
editor-pane-enabled accessor function 149
editor-pane-fixed-fill accessor function 149
editor-pane-line-wrap-face accessor function 149
editor-pane-line-wrap-marker accessor function 149
editor-pane-native-blink-rate function 163
editor-pane-selected-text generic function 163
editor-pane-selected-text-p generic function 164
editor-pane-stream function 164
editor-pane-text accessor function 149
editor-pane-wrap-style accessor function 149
editor-window generic function 165
element class 165
element-container function 172
element-interface accessor function 167
element-interface-for-callback generic function 173
element-parent accessor function 167
element-screen function 174
element-widget-name accessor function 167
ellipse class 174
:enabled initarg 26, 148, 181, 337, 367, 551, 584, 631
:enabled-function initarg 337, 631
:enabled-function-for-dialog initarg 332
:enabled-positions initarg 368
:enabled-slot initarg 337
:enable-pointer-documentation initarg 228
:enable-tooltips initarg 228
:end-x initarg 271
:end-y initarg 271
ensure-<command> function 835
ensure-area-visible generic function 174
ensure-gdiplus function 716
ensure-interface-screen function 175
Escape key 415
event handler
key strokes 374
mouse click 374
mouse gestures 374
mouse move 374
execute-with-interface function 176
execute-with-interface-if-alive function 177
exit-confirm function 178
exit-dialog function 179
expandable-item-pinboard-object class 180
:expandp-function initarg 640
:extend-callback initarg 38
extended-selection-tree-view class 180
:external-border initarg 229
external-image class 717
external-image-color-table function 718
externalize-and-write-image function 719
externalize-image function 721
:external-max-height initarg 167, 405
:external-max-width initarg 167, 405
:external-min-height initarg 166, 405
:external-min-width initarg 166, 405
F
f2pi constant 722
:file-completion initarg 585
:filename initarg 488
filled accessor function 174, 478
:filled initarg 174, 478
fill-style graphics state parameter 743
:filter initarg 273
:filter-automatic-p initarg 273
:filter-callback initarg 273
:filter-change-callback-p initarg 273
:filter-help-string initarg 273
filtering-layout class 181
filtering-layout-matches-text accessor function 182
filtering-layout-match-object-and-exclude-p function 185
filtering-layout-state accessor function 182
:filter-matches-title initarg 273
:filter-short-menu-text initarg 273
find-best-font function 723
find-graph-edge generic function 185
find-graph-node generic function 186
finding panes
interfaces 93
find-interface generic function 187
find-matching-fonts function 724
find-pane 93
find-string-in-collection generic function 188
:fit-size-to-children initarg 573
:fixed-fill initarg 149
:flatp initarg 620
focus
keyboard input on Cocoa 236
mouse events on Cocoa 237
moving to a new pane 5
setting to a pane 390, 535
:focus-callback initarg 373
font graphics state parameter 745
:font initarg 551
font type 724
font-description function 726
font-description type 725
font-description-attributes function 727
font-description-attribute-value function 727
font-dual-width-p function 726
font-fixed-width-p function 728
font-single-width-p function 729
force-screen-update function 188
force-update-all-screens function 189
foreground graphics state parameter 742
:foreground initarg 551
foreign-owned-interface class 189
form-layout class 190
form-title-adjust accessor function 190
form-title-gap accessor function 190
form-vertical-adjust accessor function 190
form-vertical-gap accessor function 190
fpi constant 730
fpi-by-2 constant 730
frame 616
free-image function 730
free-image-access function 731
free-metafile function 191
free-sound function 192
:from initarg 198
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 680
apply-in-pane-process 8
apply-in-pane-process-if-alive 9
apply-rotation 680
apply-rotation-around-point 681
apply-scale 682
apply-translation 683
apropos-color-alias-names
apropos-color-names 824
apropos-color-spec-names 825
attach-interface-for-callback 11
attach-simple-sink 12
attach-sink 13
augment-font-description 683
beep-pane 14
can-use-metafile-p 42
capi-object-property 43
choice-selected-item-p 52
choice-update-item 54
clear-external-image-conversions 684
clear-graphics-port 685
clear-graphics-port-state 685
clear-rectangle 686
clipboard-empty 57
cocoa-view-pane-view 63
color-<component> 826
color-level 826, 828
color-model 829
colors= 830
color-with-alpha 829
component-name 76
compress-external-image 687
compute-char-extents 687
confirmer-pane 78
confirm-quit 76
confirm-yes-or-no 78
contain 79
convert-color 831
convert-external-image 688
convert-relative-position 80
convert-to-font-description 689
convert-to-screen 81
copy-area 690
copy-external-image 691
copy-pixels 692
convert-transform 693
create-pixmap-port 693
current-dialog-handle 85
current-pointer-position 87
current-popup 88
current-printer 89
default-library 90
define-color-alias 832
define-font-alias 695
delete-color-translation 834
destroy-pixmap-port 696
detach-simple-pixmap-port 103
detach-sink 104
display 104
display-message 110
display-message-for-pane 111
display-message-on-screen 109
display-pane-selected-text 109
display-pane-selection 113
display-pane-selection-p 114
display-popup-menu 113
display-replacable-dialog 116
dither-color-spec 696
docking-layout-pane-docked-p 122
docking-layout-pane-visible-p 122
drag-pane-object 131
draw-arc 697
draw-arcs 698
draw-character 699
draw-circle 700
draw-ellipse 701
draw-image 702
draw-line 704
draw-lines 705
draw-metafile 133
draw-metafile-to-image 134
draw-path 705
draw-point 709
draw-points 710
draw-polygon 710
draw-polygons 711
draw-rectangle 712
draw-rectangles 713
draw-string 714
drop-object-allows-drop-effect-p 139
drop-object-collection-effect-p 139
drop-object-collection-index 139
drop-object-collection-item 141
drop-object-drop-effect 142
try 396
pane-supports-menus-with-images 398
pixblt 777
play-sound 400
popup-confirmers 413
port-graphics-state 779
port-height 780
port-string-height 780
port-string-width 781
postmultiply-transforms 782
premultiply-transforms 782
print-dialog 427
print-editor-buffer 428
printer-configuration-dialog 431
printer-metrics-device-height 432
printer-metrics-device-width 432
printer-metrics-dpi-x 433
printer-metrics-dpi-y 433
printer-metrics-height 433
printer-metrics-left-margin 433
printer-metrics-max-height 433
printer-metrics-max-width 433
printer-metrics-min-left-margin 433
printer-metrics-min-top-margin 433
printer-metrics-paper-height 433
printer-metrics-paper-width 433
printer-metrics-top-margin 433
printer-metrics-width 433
printer-port-handle 434
printer-port-supports-p 434
print-file 429
print-rich-text-pane 430
print-text 431
process-pending-messages 436
prompt-for-color 437
prompt-for-confirmation 438
prompt-for-directory 440
prompt-for-file 442
prompt-for-files 445
prompt-for-font 446
prompt-for-form 447
prompt-for-forms 449
prompt-for-integer 450
prompt-for-items-from-list 451
prompt-for-number 452
prompt-for-string 453
prompt-for-symbol 455
prompt-for-value 457
prompt-with-list 458
prompt-with-list-non-focus 461
prompt-with-message 466
quit 60
quit-interface 470
raise-interface 474
range-set-sizes 476
read-and-convert-external-image 783
read-color-db 842
read-external-image 784
read-sound-file 477
rectangle-union 788
redisplay-menu-bar 479
redraw-pinboard-layout 480
redraw-pinboard-object 481
register-image-load-function 790
register-image-translation 791
remove-capi-object-property 482
replace-dialog 484
reset-image-translation-table 792
reuse-interfaces-p 487
rich-text-pane-character-format 490
rich-text-pane-operation 491
rich-text-pane-paragraph-format 495
rich-text-version 496
screen-active-interface 501
screen-active-p 501
screen-internal-geometries 503
screen-internal-geometry 505
screen-logical-resolution 502
screen-monitor-geometries 504
screens 506
selection 512
selection-empty 514
separation 793
set-application-interface 514
set-clipboard 516
set-composition-placement 518
set-confirm-quit-flag 519
set-default-editor-pane-blink-rate 519
set-default-image-load-function 793
set-default-interface-prefix-suffix 520
set-default-use-native-input-method 522
set-drop-object-supported-formats 523
set-editor-parenthesis-colors 525
set-geometric-hint 526
set-graphics-port-coordinates 794
set-graphics-state 795
set-hint-table 526
set-interactive-break-gestures 528
set-list-panel-keyboard-search-reset-time 529
set-object-automatic-resize 530
set-printer-metrics 541
set-printer-options 542
set-rich-text-pane-character-format 535
set-rich-text-pane-paragraph-format 538
set-selection 540
set-text-input-pane-selection 522
show-interface 548
show-pane 549
simple-pane-handle 561
simple-print-port 565
sorted-object-sorted-by 571
sort-object-items-by 569
start-gc-monitor 572
stop-gc-monitor 574
stop-sound 575
tab-layout-panes 582
tab-layout-visible-child 582
text-input-pane-append-recent-items 599
text-input-pane-complete-text 604
text-input-pane-copy 605
text-input-pane-cut 605
text-input-pane-delete 606
text-input-pane-delete-recent-items 599
text-input-pane-in-place-complete 606
text-input-pane-paste 607
text-input-pane-prepend-recent-items 600
text-input-pane-recent-items 601
text-input-pane-replace-recent-items 602
text-input-pane-selected-text 113, 608
text-input-pane-selection 113, 608
text-input-pane-selection-p 114, 609
text-input-pane-set-recent-items 602
transform-area 796
transform-distance 796
transform-distances 797
transform-is-rotated 797
transform-point 798
transform-points 798
transform-rect 799
tree-view-ensure-visible 649
tree-view-item-checkbox-status 650
tree-view-item-children-checkbox-status 651
unconvert-color 843
undefine-font-alias 800
uninstall-postscript-printer 654
unit-transform-p 801
unmap-typeout 655
untransform-distance 802
untransform-distances 803
untransform-point 803
untransform-points 804
update-all-interface-titles 665
update-pinboard-object 656
update-screen-interface-titles 657
update-toolbar 658
virtual-screen-geometry 658
wrap-text 675
wrap-text-for-pane 676
write-external-image 820

G
:gap initarg 73, 498
generic functions
accepts-focus-p 4
append-items 7
browser-pane-busy 23
browser-pane-go-back 23
browser-pane-go-forward 23
browser-pane-navigate 23
browser-pane-property-get 25
browser-pane-property-put 25
browser-pane-refresh 23
browser-pane-stop 23
calculate-constraints 36
calculate-layout 37
call-editor 41
choice-selected-item 50
choice-selected-items 52
collapse 57
collect-interfaces 64
collection-find-next-string 69
collection-find-string 70
collection-last-search 70
collection-search 71
count-collection-items 85
current-document 87
destroy 102
display-dialog 107
display-tooltip 117
draw-pinboard-object 137
draw-pinboard-object-highlighted 138
draw-pinboard-object-unhighlighted 138
drop-object-pane-x 144
drop-object-pane-y 144
drop-pane-blank-rate 158
draw-pane-selected-text 163
draw-pane-selected-text-p 164
draw-pane-window 165
element-interface-for-call-back 173
ensure-area-visible 174
find-graph-edge 185
find-graph-node 186
find-interface 187
find-string-in-collection 188
get-collection-item 192
get-horizontal-scroll-parameters 193
get-vertical-scroll-parameters 197
graph-node-children 199
graph-pane-add-graph-node 204
graph-pane-delete-object 205
graph-pane-delete-objects 205
graph-pane-delete-selected-objects 206
graph-pane-direction 207
graph-pane-select-graph-nodes 209
graph-pane-update-moved-objects 210
highlight-pinboard-object 217
interactive-pane-execute-command 226
interface-display 245
interface-editor-pane 248
interface-extend-title 248
interface-geometry 249
interface-keys-style 250
interface-match-p 253
interface-menu-groups 254
interface-preserve-state 255
interface-reuse-p 256
interpret-description 260
invalidate-rectangle 760
itemp 264
item-pane-interface-copy-object 264
list-panel-enabled 283
list-panel-filter-state 284
list-panel-unfiltered-items 288
locate-interface 300
make-container 302
make-image-access 770
make-pane-popup-menu 309

863
manipulate-pinboard 315
map-collection-items 318
map-pane-children 319
map-pane-descendant-children 322
merge-menu-bars 340
move-line 343
non-focus-maybe-capture-gesture 351
non-focus-terminate 353
over-pinboard-object-p 384
pane-adjusted-offset 386
pane-adjusted-position 387
pane-got-focus 390
pane-has-focus-p 390
pane-initial-focus 391
pane-interface-copy-object 392
pane-interface-copy-p 392
pane-interface-cut-object 392
pane-interface-cut-p 392
pane-interface-deselect-all 392
pane-interface-deselect-all-p 392
pane-interface-paste-object 392
pane-interface-paste-p 392
pane-interface-select-all 392
pane-interface-select-all-p 392
pane-interface-undo 392
pane-popup-menu-items 393
pane-string 397
parse-layout-descriptor 398
pinboard-object-at-position 408
pinboard-object-graphics-arg 409
pinboard-object-overlap-p 410
pinboard-pane-position 411
pinboard-pane-size 412
port-drawing-mode-quality-p 778
print-capi-button 425
print-collection-item 425
redisplay-collection-item 478
redisplay-interface 479
reinitialize-interface 481
remove-items 483
replace-items 484
report-active-component-failure 485
scroll 507
scroll-if-not-visible-p 511
search-for-item 512
set-button-panel-enabled-items 516
set-display-pane-selection 522
set-horizontal-scroll-parameters 527
set-pane-focus 535
set-scroll-position 508
set-scroll-range 528, 546
set-text-input-pane-selection 544
set-top-level-interface-geometry 544
set-vertical-scroll-parameters 546
simple-pane-visible-height 562
simple-pane-visible-size 562
simple-pane-visible-width 563
sorted-object-sort-by 570
switchable-layout-switchable-children 577
top-level-interface 631
top-level-interface-display-state 632
top-level-interface-geometry 633
top-level-interface-geometry-key 634
top-level-interface-p 636
top-level-interface-save-geometry-p 636
tree-view-expanded-p 650
tree-view-update-an-item 652
tree-view-update-item 652
unhighlight-pinboard-object 653
update-interface-title 656
validate-rectangle 805
geometry slots
get-font-width function 737
get-graphics-state function 738
get-horizontal-scroll-parameters generic function 193
get-origin function 738
get-page-area function 194
get-pane 93
get-printer-metrics function 195
get-scroll-position function 196
get-string-extent function 739
get-transform-scale function 740
get-vertical-scroll-parameters generic function 197
graph-edge class 198
graph-edge-from accessor function 198
graph-edge-to accessor function 198
:graphics-args initarg 404
:graphics-options initarg 372
graphics-port-background function 740
graphics-port-font function 740
graphics-port-foreground function 740
graphics-port-transform function 740
graphics-state structure class 741
graphics-state-background function 746
graphics-state-compositing-mode function 746
graphics-state-dash function 746
graphics-state-dashed function 746
graphics-state-fill-style function 746
graphics-state-font function 746
graphics-state-foreground function 746
graphics-state-line-end-style function 746
graphics-state-line-joint-style function 746
graphics-state-mask function 746
graphics-state-mask-transform function 746
graphics-state-mask-x function 746
graphics-state-mask-y function 746
graphics-state-operation function 746
**graphics-state-pattern** function 746
**graphics-state-scale-thickness** function 746
**graphics-state-shape-mode** function 746
**graphics-state-stipple** function 746
**graphics-state-text-mode** function 746
**graphics-state-transform** function 746
**graph-node** class 199
**graph-node-children** generic function 199
**graph-node-height** accessor function 199
**graph-node-out-edges** accessor function 199
**graph-node-width** accessor function 199
**graph-node-x** accessor function 199
**graph-node-y** accessor function 199
**graph-object** class 200
**graph-object-element** accessor function 200
**graph-object-object** accessor function 200
**graph-pane** class 200
**graph-pane-add-graph-node** generic function 204
**graph-pane-delete-object** generic function 205
**graph-pane-delete-selected-objects** generic function 205
**graph-pane-direction** generic function 207
**graph-pane-edges** function 208
**graph-pane-layout-function** accessor function 201
**graph-pane-nodes** function 208
**graph-pane-object-at-position** function 209
**graph-pane-roots** accessor function 201
**graph-pane-select-graph-nodes** generic function 209
**graph-pane-update-moved-objects** generic function 210
**grid-layout** class 211
**groupbox** 616
GTK+ resources 83, 168, 529

**H**

:has-root-line initarg 640
:has-title-column-p initarg 211
:head initarg 10
:head-breadth initarg 10
:head-direction initarg 10
:header-args initarg 345
:head-graphics-args initarg 10
:head-length initarg 10
%height% geometry slot 667
;height initarg 500
help
  context help 237
  help-callback 233
:help item in :buttons initarg 593
:help-callback initarg 228, 229
help-key accessor function 66, 167, 332, 623
:help-key initarg 66, 166, 331, 623
:help-keys initarg 31
:help-string initarg 182
hide-interface function 216
hide-pane function 216
highlight-pinboard-object generic function 217
:highlight-style initarg 401
:hist-addtofavorites image symbol 281, 624, 644
:hist-back image symbol 281, 624, 644
:hist-favorites image symbol 281, 624, 644
:hist-forward image symbol 281, 624, 644
:hist-viewport image symbol 281, 624, 644
:horizontal-scroll initarg 510, 551
HWND 83, 561

**I**

:iconify-callback initarg 228
:ignore-file-suffixes initarg 585
image class 750
image identifiers
function 250
```
interface-match-p generic function 253
interface-menu-bar-items accessor function 230
interface-menu-groups generic function 254
interface-message-area accessor function 230, 239
interface-override-cursor accessor function 230
interface-pathname accessor function 230
interface-reuse-p generic function 256
```

:interfaces class option 101
```
:interfaces initarg 500
interface-title accessor function 230
interface-toolbar-items accessor function 230
interface-toolbar-state function 257
interface-toolbar-styles accessor function 230
interface-tooltip-enabled accessor function 230
internal scrolling 378
```

:internal-border initarg 551
```
:internal-max-height initarg 167, 406
:internal-max-width initarg 167, 406
:internal-min-height initarg 167, 405
:internal-min-width initarg 167, 405
```

:interpret-description generic function 260
```
Interrupt playing a MIDI file 575
invalidate-pane-constraints function 261
invalidate-rectangle generic function 760
invert-transform function 761
invoke-command function 261
invoke-untranslated-command function 262
item class 262
item-collection accessor function 263
item-data accessor function 263
itemp generic function 264
item-pane-interface-copy-object generic function 264
item-pinboard-object class 266
item-print-function accessor function 263
```

:items initarg 66, 118, 180, 324, 329, 578, 649
```
:items-count-function initarg 66, 180, 649
:items-selected accessor function 263
:items-function initarg 324, 329
:items-get-function initarg 66, 180, 649
:items-map-function initarg 66, 180, 649
```
item-text accessor function 263
```

K
```
:keep-selection-p initarg 47
```
key press event handler 374
```
:keyboard-search-callback initarg 273
```
key-function initarg 578
key-press events 374
```

L
```
labelled-arrow-pinboard-object class 267
labelled-line-pinboard-object class 267
labelled-line-text-background accessor function 267
labelled-line-text-foreground accessor function 267
```
```
:label-style initarg 182
```
:large-image-height initarg 290
```
```
:large-image-width initarg 290
layout class 268
:layout initarg 227
:layout-args initarg 31
:layout-class initarg 31
layout-description accessor function 269
*layout-divider-default-size* 74, 498
:layout-function initarg 201
layout-ratios accessor function 74, 498
:layouts class option 93
layout-x-adjust accessor function 677
:layout-x-adjust initarg 201
layout-x-gap accessor function 211
layout-x-ratios accessor function 211
layout-y-adjust accessor function 677
:layout-y-adjust initarg 201
layout-y-gap accessor function 211
layout-y-ratios accessor function 211
:leaf-node-p-function initarg 639
line-end-style graphics state parameter 744
line-joint-style graphics state parameter 744
line-pinboard-object class 270
line-pinboard-object-coordinates function 271
:line-size initarg 509
:line-wrap-face initarg 149
:line-wrap-marker initarg 149
LispWorks as ActiveX control 100, 356
list-all-font-names function 761
listener-pane class 294
listener-pane-insert-value function 295
list-known-image-formats function 762
list-panel class 272
list-panel-enabled generic function 283
list-panel-filter-state generic function 284
list-panel-image-function accessor function 274
list-panel-items-and-filter function 285
list-panel-keyboard-search-callback accessor function 274
list-panel-right-click-selection-behavior accessor function 274
list-panel-search-with-function function 286
list-panel-state-image-function accessor function 274
list-panel-unfiltered-items generic function 288
list-view class 288
list-view-auto-arrange-icons accessor function 290
list-view-auto-reset-column-widths accessor function 290
list-view-columns accessor function 290
list-view-image-function accessor function 290
list-view-state-image-function accessor function 290
list-view-subitem-function accessor function 290
list-view-subitem-print-functions accessor function 290
list-view-view accessor function 290
load-color-database function 839
load-cursor function 295
load-icon-image function 763
load-image function 765
load-sound function 299
locate-interface generic function 300
lookup pane interface 93
lookup-pane 93
lower-interface function 301
M
Mac OS X Dock 58
macros
define-color-models 833
define-command 90
define-interface 92
define-layout 98
define-menu 99
define-ole-control-component 100
display-errors 110
rectangle-bind 785
rectangle-bottom 786
rectangle-height 786
rectangle-left 787

869
rectangle-right 787
rectangle-top 788
rectangle-width 789
rect-bind 790
undefined-menu 653
union-rectangle 800
unless-empty-rect-bind 802
with-atomic-redisplay 659
with-busy-interface 660
with-dialog-results 661
with-dither 806
with-document-pages 663
with-external-metafile 664
with-geometry 667
with-graphics-mask 806
with-graphics-post-transformation 808
with-graphics-rotation 809
with-graphics-scale 810
with-graphics-state 810
with-graphics-transform 812
with-graphics-transform-reset 813
with-graphics-translation 814
with-internal-metafile 669
with-inverse-graphics 815
with-output-to-printer 671
without-relative-drawing 815
with-page 672
with-page-transform 673
with-pixmap-graphics-port 816
with-print-job 673
with-random-typeout 674
with-transformed-area 817
with-transformed-point 818
with-transformed-points 819
with-transformed-rect 819
make-container generic function 302
make-dither function 767
make-docking-layout-controller function 303
make-font-description function 768
make-foreign-owned-interface function 303
make-general-image-set function 305
make-graphics-state function 769
make-gray function 839
make-hsv function 840
make-icn-resource-image-set function 306
make-image function 769
make-image-access generic function 770
make-image-from-port function 771
make-image-locator function 307
make-menu-for-pane function 307
make-pane-popup-menu generic function 309
make-resource-image-set function 311
make-rgb function 841
make-scaled-general-image-set function 312
make-scaled-image-set function 313
make-sorting-description function 314
make-sub-image function 772
make-transform function 773
manipulate-pinboard generic function 315
map-collection-items generic function 318
map-pane-children generic function 319
map-pane-descendant-children generic function 322
map-typeout function 323
mask graphics state parameter 744
mask-transform graphics state parameter 745
mask-x graphics state parameter, deprecated 745
mask-y graphics state parameter, deprecated 745
:matches-title initarg 181
:max-characters initarg 584
%max-height% geometry slot 668
*maximum-moving-objects-to-track-edges* variable 323
:maximum-recent-items initarg 586
%max-width% geometry slot 668
MDI 81, 87, 124
menu class 324
:menu initarg 424
:menu-bar class option 93
:menu-bar-items initarg 227
menu-component class 328
:menu-function initarg 424
menu-image-function accessor function 324
menu-item class 331
menu-items accessor function 324
menu-object class 337
menu-enabled accessor function 337
menu-popup-callback accessor function 337
:menus class option 93
menu-title accessor function 613
menu-title-function accessor function 613
merge-font-descriptions function 774
merge-menu-bars generic function 340
:message initarg 615
:message-area initarg 228
:message-callback initarg 58
:message-gap initarg 615
message-pane class 342
MIDI files
interrupting 575
%min-height% geometry slot 668
%min-width% geometry slot 667
:mnemonic initarg 26, 32, 324, 331
:mnemonic-escape initarg 27, 32, 324, 332
:mnemonic-text initarg 27, 32
:mnemonic-title initarg 32, 324, 332, 615
modal dialogs 108, 417, 661
modify-editor-pane-buffer function 342
mono-screen class 343
Motif resources 168
mouse clicks 374
mouse coordinates 87
mouse events 374
mouse position 87
move-line generic function 343
multi-column-list-panel class 344
multi-line-text-input-pane class 349
Multiple Document Interface 81, 87, 124
multiple-selection interaction style 27
N
:name initarg 43
:navigate-complete-callback initarg 15
:navigate-error-callback initarg 16
:navigation-callback initarg 585
New in LispWorks 6.1
all-button argument for prompt-with-list 458
:alternating-background initarg for list-panel 273
:alternative-action-call-back initarg for callbacks 38
apply-in-pane-process-if-alive 9
apply-rotation-around-point 681
:auto-arrange-icons initarg for list-view 289
:automatic-resize initarg for pinboard-object 404
:automatic-resize initarg for simple-pane 552
browser-pane 15
buttons argument for prompt-with-list 458
callbacks argument for prompt-with-list 458
can-use-metafile-p 42
:change-callback initarg for text-input-range 610
changed behavior of set-object-automatic-resize 534
:color-function initarg for list-panel on Cocoa 272
color-with-alpha 829
compositing-mode graphics state parameter 745
:composition-callback initarg for output-pane 373
:composition-face initarg for editor-pane 149
copy-area 690
:delete-item-callback initarg for tree-view 640
display-pane-selected-text 113
display-pane-selection 113
display-pane-selection-p 114
display-tooltip supported on GTK+ 117
:documentation option in define-interface 93
drag and drop images 524, 750
:drag-image initarg for interface 229
:drawing-mode argument for simple-print-port 566
:drawing-mode argument for with-print-job 674
:drawing-mode argument to create-pixmap-port 694
:drawing-mode argument to with-external-metafile 665
:drawing-mode argument to with-internal-metafile 670
:drawing-mode argument to with-pixmap-graphics-port 816
:drawing-mode initarg for editor-pane 154
:drawing-mode initarg for graph-pane 202
:drawing-mode initarg for output-pane 372
:drawing-mode initarg for pin-board-layout 403
draw-path 705
:drop-callback supported for list-panel and tree-view on Cocoa and GTK+ 552
*editor-pane-composition-selected-range-face-plist* 160
editor-pane-default-composition-callback 160
*editor-pane-default-composition-face* 162
:enhanced-gdi metafile format on Windows 664, 670
:enhanced-plus metafile format on Windows 664, 670
Example illustrating display of fractional and logarithmic values in slider 569
externalize-and-write-image 719
:finished-launching message callback 60
Graphics state mask can be a path 807
image-access-height 750
image-access-width 750
image-count argument for make-scaled-general-image-set 312
:image-function initarg for tablayout 578
:image-lists initarg for option-pane 368
:image-lists initarg for tablayout 578
Initargs for handling images in list-panel 273
:initial-constraints initarg for element 166
interface-document-modified-p 247
interface-drag-image 230
interface-pathname 230
item-pane-interface-copy-object 264
:keyboard-search-callback initarg for list-panel 273
list-known-image-formats 762
list-panel supports images 272
list-panel-image-function 274
list-panel-search-with-function 286
list-panel-state-image-function 274
list-view-auto-arrange-icons 290
:mask-transform argument in with-graphics-mask 807
mask-transform graphics state parameter 745
:maximum-recent-items initarg for text-input-pane 586
none-button argument for prompt-with-list 458
pane-screen-internal-geometry 396
pathname initarg for interface 229
port-drawing-mode-quality-p 778
:print-function initarg for slider 566
:recent-items initarg for text-input-pane 586
:recent-items-mode initarg for text-input-pane 586
:recent-items-name initarg for
text-input-pane 586
:redo-items argument to redis-
play-menu-bar 479
screen-internal-geometries
503
screen-monitor-geometries
504
:search-field initarg for text-
input-pane 586
:selected-item-function init-
targ for toolbar-component
629
:selected-items-function init-
targ for toolbar-component
629
:selection-function initarg for
toolbar-component 629
set-composition-placement
518
set-default-use-native-
input-method 522
set-display-pane-selection
522
set-list-panel-keyboard-
search-reset-time 529
shape-mode graphics state parameter 745
:show-value-p initarg for
slider supported on Windows
567
slider-print-function 566
slider-tick-frequency 567
sorted-object-sorted-by 571
Symbols loading history images on Win-
dows 281, 624, 644
Symbols loading view images on Win-
dows 280, 624, 644
tab-layout-image-function
578
:text-background initarg for
labelled-line-pinboard-
object 267
text-input-pane-append-
recent-items 599
text-input-pane-delete-
recent-items 599
text-input-pane-prepend-
recent-items 600
text-input-pane-recent-
items 601
text-input-pane-replace-
recent-items 602
text-input-pane-set-recent-
items 602
text-mode graphics state parameter 745
:tick-frequency initarg for
slider 566
:use-native-input-method ini-
targ for output-pane 373
virtual-screen-geometry 658
:window-styles initarg for
option-pane 368
with-graphics-post-transla-
tion 808
with-graphics-transform-
reset 813
:new-window-callback initarg 15
:node-pane-function initarg 201
:node-pinboard-class initarg 201
non-focus-list-add-filter
function 351
non-focus-list-interface class
350
non-focus-list-remove-filter
function 351
non-focus-list-toggle-enable-filter
function 350
non-focus-list-toggle-filter
function 351
non-focus-maybe-capture-ges-
ture generic function 351
non-focus-terminate generic func-
tion 353
non-focus-update generic function
354
no-selection interaction style 27
:number initarg 500

O
%object% geometry slot 669
offset-rectangle function 775
:ok item in :buttons initarg 592
ok-button image identifier 594
OLE control 100, 356
OLE embedding 100, 356
ole-control-add-verbs function
354
ole-control-close-object func-
tion 355
ole-control-component class 356
ole-control-component-pane
accessor function 356
ole-control-doc class 358
ole-control-frame class 359
ole-control-i-dispatch function 360

ole-control-insert-object function 361

ole-control-ole-object function 361

ole-control-pane class 362

ole-control-pane-frame function 365

ole-control-pane-simple-sink class 365

ole-control-user-component function 366

operation graphics state parameter 742

option-pane class 367

option-pane-enabled accessor function 368

option-pane-enabled-positions accessor function 368

option-pane-image-function accessor function 368

option-pane-popup-callback accessor function 368

option-pane-separator-item accessor function 368

option-pane-visible-items-count accessor function 368

ordered-rectangle-union function 775

ordinary scrolling 378

:orientation initarg 119, 211, 476

:orientation item in :buttons initarg 594

output-pane class 371

output-pane-composition-callback accessor function 373

output-pane-create-callback accessor function 373

output-pane-destroy-callback accessor function 373

output-pane-display-callback accessor function 373

output-pane-focus-callback accessor function 373

output-pane-graphics-options accessor function 373

output-pane-input-model accessor function 373

output-pane-notify-accessor accessor function 373

output-pane-redraw-callback accessor function 373

output-pane-scroll-callback accessor function 373

over-pinboard-object-p generic function 384

:override-cursor initarg 228

P

page-setup-dialog function 385

:page-size initarg 509

pane-adjusted-offset generic function 386

pane-adjusted-position generic function 387

:pane-can-scroll initarg 372

pane-close-display function 388

pane-descendant-child-with-focus function 389

:pane-function initarg 356

pane-got-focus generic function 390

pane-has-focus-p generic function 390

pane-initial-focus generic function 391

pane-interface-copy-object generic function 392

pane-interface-copy-p generic function 392

pane-interface-cut-object generic function 392

pane-interface-cut-p generic function 392

pane-interface-deselect-all generic function 392

pane-interface-deselect-all-p generic function 392

pane-interface-paste-object generic function 392

pane-interface-paste-p generic function 392

pane-interface-select-all generic function 392

pane-interface-select-all-p generic function 392

pane-interface-undo generic function 392

pane-interface-undo-p generic function 392

pane-layout accessor function 32, 230

:pane-menu initarg 552

pane-popup-menu-items generic function 393

:panes class option 93
*printer-search-path* variable 435
print-file function 429
:print-function initarg 66, 263, 566, 578
print-rich-text-pane function 430
print-text function 431
process
  CAPI 105
  Cocoa Event Loop 106
process-pending-messages function 436
progress-bar class 437
prompt-for-color function 437
prompt-for-confirmation function 438
prompt-for-directory function 440
prompt-for-file function 442
prompt-for-files function 445
prompt-for-font function 446
prompt-for-form function 447
prompt-for-forms function 449
prompt-for-integer function 450
prompt-for-items-from-list function 451
prompt-for-number function 452
prompt-for-string function 453
prompt-for-symbol function 455
prompt-for-value function 457
prompt-with-list function 458
prompt-with-list-non-focus function 461
prompt-with-message function 466
:protected-callback initarg 488
push-button class 467
push-button-panel class 469
Q
quit function 60
quit-interface function 470
R
radio-button class 472
radio-button-panel class 473
raise-interface function 474
range-callback accessor function 476
range-end accessor function 476
range-orientation accessor function 476
range-pane class 475
range-set-sizes function 476
range-slug-end accessor function 476
range-slug-start accessor function 476
range-start accessor function 476
:ratios initarg 73, 498
read-and-convert-external-image function 783
read-color-db function 842
read-external-image function 784
read-sound-file function 477
:recent-items initarg 586
:recent-items-mode initarg 586
:recent-items-name initarg 586
rectangle class 478
rectangle-bind macro 785
rectangle-bottom macro 786
rectangle-height macro 786
rectangle-left macro 787
rectangle-right macro 787
rectangle-top macro 788
rectangle-union function 788
rectangle-width macro 789
rect-bind macro 790
red Close button
  on Cocoa 247
redisplay-collection-item
generic function 478
redisplay-interface generic function 479
redisplay-menu-bar function 479
redraw-pinboard-layout function 480
redraw-pinboard-object function 481
register-image-load-function
  function 790
register-image-translation
  function 791
reinitialize-interface generic function 481
:remapped initarg 623
remove-capi-object-property
  function 482
remove-items generic function 483
replace-dialog function 484
replace-items generic function 484
report-active-component-failure generic function 485
reset-image-translation-table function 792
resetable
dialogs 237
elements 170
windows 231
:resize-callback initarg 373
resizing 170, 231, 237
resolution
of display 502
scale on printer 195
:retain-expanded-nodes initarg 639
:retract-callback initarg 28, 38
Return key 415
reuse-interfaces-p function 487
rich-text-pane class 487
rich-text-pane-change-callback accessor function 488
rich-text-pane-character-format function 490
rich-text-pane-limit accessor function 488
rich-text-pane-operation function 491
rich-text-pane-paragraph-format function 495
rich-text-pane-text accessor function 488
rich-text-version function 496
right-angle-line-pinboard-object class 496
:right-click-extended-match initarg 640
:right-click-selection-behavior initarg 272
:roots initarg 201, 639
row-layout class 497
row-layout-divider 498
:rows initarg 211

S
:save-name initarg 362
scale
for a printer 195
scale-thickness graphics state parameter 744
scaling
while printing 673
screen
usable region of 505
screen class 499
screen-active-interface function 501
screen-active-p function 501
screen-depth accessor function 500
screen-height accessor function 500
screen-height-in-millimeters accessor function 500
screen-interfaces accessor function 123, 500
screen-internal-geometries function 503
screen-internal-geometry function 505
screen-logical-resolution function 502
screen-monitor-geometries function 504
screen-number accessor function 500
screens function 506
screen-width accessor function 500
screen-width-in-millimeters accessor function 500
scroll generic function 507
scroll-bar class 509
scroll-bar-line-size accessor function 509
scroll-bar-page-size accessor function 509
scroll-bar-page-size accessor function 509
:scroll-callback initarg 372
%scroll-height% geometry slot 668
%scroll-horizontal-page-size% geometry slot 668
%scroll-horizontal-slug-size% geometry slot 668
%scroll-horizontal-step-size% geometry slot 668
%scroll-if-not-visible-p generic function 511
:scroll-if-not-visible-p initarg 552
scrolling
built-in 196
internal 378
ordinary 378
%scroll-start-x% geometry slot 668
%scroll-start-y% geometry slot 668
%scroll-vertical-page-size% geometry slot 668
%scroll-vertical-slug-size% geometry slot 668
%scroll-vertical-step-size% geometry slot 668
%scroll-width% geometry slot 668
simple-pane-drop-callback accessor function 552
simple-pane-enabled accessor function 182, 552, 631
simple-pane-font accessor function 552
simple-pane-foreground accessor function 552
simple-pane-handle function 561
simple-pane-horizontal-scroll accessor function 552
simple-pane-scroll-callback accessor function 552
simple-pane-vertical-scroll accessor function 552
simple-pane-visible-border accessor function 552
simple-pane-visible-height generic function 562
simple-pane-visible-size generic function 562
simple-pane-visible-width generic function 563
simple-pinboard-layout class 564
simple-print-port function 565
single-selection interaction style 27
:sinks initarg 363
slider class 566
slider-print-function accessor function 566
slider-show-value-p accessor function 567
slider-start-point accessor function 567
slider-tick-frequency accessor function 567
:slug-end initarg 475
:slug-start initarg 475
:small-image-height initarg 290
:small-image-width initarg 290
sorted-object class 570
sorted-object-sort-by generic function 570
sorted-object-sorted-by function 571
sort-object-items-by function 569
:source-interfaces class option 101
special slots
car container 124
windows-menu 124
standard image symbol
:std-copy 280, 624, 643
:std-cut 280, 624, 643
:std-delete 280, 624, 643
:std-file-new 280, 624, 644
:std-file-open 280, 624, 644
:std-file-save 280, 624, 644
:std-find 280, 624, 644
:std-help 280, 624, 644
:std-paste 280, 624, 643
:std-print 280, 624, 644
:std-print-pre 280, 624, 644
:std-properties 280, 624, 644
:std-redo 280, 624, 643
:std-replace 280, 624, 644
:std-undo 280, 624, 643
:start initarg 475, 610
start-gc-monitor function 572
:start-point initarg 566
:start-x initarg 271
:start-y initarg 271
:state-image-function initarg 273, 289, 641
:state-image-height initarg 274, 290, 641
:state-image-width initarg 274, 290, 641
static-layout class 573
:status-text-change-callback initarg 15
:std-copy image symbol 280, 643
:std-cut image symbol 280, 643
:std-delete image symbol 280, 643
:std-file-new image symbol 280, 644
:std-file-open image symbol 280, 644
:std-file-save image symbol 280, 644
:std-find image symbol 280, 644
:std-help image symbol 280, 644
:std-paste image symbol 280, 643
:std-print image symbol 280, 644
:std-print-pre image symbol 280, 644
std-properties
  image symbol 280, 644
std-redo
  image symbol 280, 643
std-replace
  image symbol 280, 644
std-undo
  image symbol 280, 643
stipple graphics state parameter 743
stop-gc-monitor function 574
stop-sound function 575
stream initarg 72
stretch-text-p initarg 620
structure classes
  graphics-state 741
  printer-metrics 432
subitem-function initarg 289
subitem-print-functions initarg 289
switchable-layout class 575
switchable-layout-combine-child-constraints accessor function 575
switchable-layout-switchable-children generic function 577
switchable-layout-visible-child accessor function 575
T
tab-layout class 577
tab-layout-combine-child-constraints accessor function 578
tab-layout-image-function accessor function 578
tab-layout-panes function 582
tab-layout-visible-child function 582
tab-layout-visible-child-function accessor function 578
tabstops 4
text-function initarg 66
text initarg 112, 148, 181, 263, 488, 584, 610, 612
text-background initarg 267
text-change-callback initarg 586
text-foreground initarg 267
text-input-choice class 583
text-input-pane class 584
text-input-pane-append-recent-items function 599
text-input-pane-before-change-callback accessor function 597
text-input-pane-buttons-enabled accessor function 586
text-input-pane-cancel function 586
text-input-pane-caret-position accessor function 586
text-input-pane-change-callback accessor function 586
text-input-pane-complete-text function 604
text-input-pane-completion-function accessor function 586
text-input-pane-confirm-change-function accessor function 586
text-input-pane-copy function 605
text-input-pane-cut function 605
text-input-pane-delete function 606
text-input-pane-delete-recent-items function 599
text-input-pane-extend-callback accessor function 586
text-input-pane-enabled accessor function 586
text-input-pane-in-place-complete function 606
text-input-pane-max-characters accessor function 586
text-input-pane-navigation-callback accessor function 586
text-input-pane-paste function 607
text-input-pane-prepend-recent-items function 600
text-input-pane-recent-items function 601
text-input-pane-replace-recent-items function 602
text-input-pane-selected-text function 113, 608
text-input-pane-selection function 113, 608
text-input-pane-selection-p function 114, 609
text-input-pane-set-recent-
transform-is-rotated function 797
transform-point function 798
transform-points function 798
transform-rect function 799
:transparency initarg 229
tree-view class 639
tree-view-action-callback-expand-p accessor function 642
tree-view-checkbox-change-callback accessor function 642
tree-view-checkbox-child-function function 642
tree-view-checkbox-initial-status accessor function 642
tree-view-checkbox-next-map accessor function 642
tree-view-checkbox-parent-function accessor function 642
tree-view-checkbox-status accessor function 642
tree-view-children-function accessor function 642
tree-view-ensure-visible function 649
tree-view-expanded-p generic function 650
.tree-view-expandp-function accessor function 642
tree-view-has-root-line accessor function 642
tree-view-image-function accessor function 642
tree-view-item-checkbox-status function 650
tree-view-item-children-checkbox-status function 651
tree-view-leaf-node-p-function accessor function 642
tree-view-retain-expanded-nodes accessor function 642
tree-view-right-click-extended-match accessor function 642
tree-view-roots accessor function 642
tree-view-state-image-function accessor function 657
.tree-view-update-an-item generic function 652
.tree-view-update-item generic function 652
:type initarg 497
types
font 724
font-description 725
transform 795
U
unconvert-color function 843
undefine-font-alias function 800
undefine-menu macro 653
unhighlight-pinboard-object
    generic function 653
.uniform-size-p initarg 73, 498
uninstall-postscript-printer
    function 654
union-rectangle macro 800
*unit-transform* variable 801
unit-transform-p function 801
unless-empty-rect-bind macro 802
unmap-typeout function 655
untransform-distance function 802
untransform-distances function 803
untransform-point function 803
untransform-points function 804
update-all-interface-titles
    function 655
.update-commands-callback initarg 15
update-interface-title generic function 656
update-pinboard-object function 656
*update-screen-interfaces-hooks* variable 657
update-screen-interfaces-hooks* function 657
update-toolbar function 658
:url initarg 16
:use-images initarg 274, 641
:use-large-images initarg 289
:use-native-input-method initarg 155, 373
:user-component initarg 362
:use-small-images initarg 290
:use-state-images initarg 274, 290,
validate-rectangle generic function 805

variables
*color-database* 827
*default-editor-pane-line-wrap-marker* 89
*default-image-translation-table* 695, 758
*echo-area-cursor-inactive-style* 145
*editor-cursor-active-style* 147
*editor-cursor-color* 146
*editor-cursor-inactive-style* 148
*editor-pane-composition-selected-range-face-plist* 160
*editor-pane-default-composition-face* 162
*maximum-moving-objects-to-track-edges* 323
*ppd-directory* 424
*printer-search-path* 435
*unit-transform* 801
*update-screen-interfaces-hooks* 657
:vertical-adjustment initarg 190
:vertical-gap initarg 190
:vertical-scroll initarg 510, 551
:view initarg 289
:view-class initarg 62
:view-details image symbol 280, 624, 644
:view-large-icons image symbol 280, 624, 644
:view-list image symbol 280, 624, 644
:view-net-connect image symbol 280, 624, 644
:view-net-disconnect image symbol 280, 624, 644
:view-new-folder image symbol 280, 624, 644
:view-parent-folder image symbol 280, 624, 644
:view-small-icons image symbol 280, 624, 644
:view-sort-date image symbol 280, 624, 644
:view-sort-name image symbol 280, 624, 644
:view-sort-size image symbol 280, 624, 644
:view-sort-type image symbol 280, 624, 644
virtual-screen-geometry function 658
:visible-border initarg 551
:visible-child initarg 575
:visible-child-function initarg 578
:visible-items-count initarg 367, 583
:visible-max-height initarg 167, 405
:visible-max-width initarg 167, 405
:visible-min-height initarg 167, 405
:visible-min-width initarg 167, 405

W
WAV sound files 299
:widget-name initarg 166
%width% geometry slot 667
:width initarg 500
window handle 85, 561
window title removal 235
window-modal dialogs 108, 417, 661
Windows history image symbol
:hist-addtoworkspace 281, 624, 644
:hist-back 281, 624, 644
:hist-favorites 281, 624, 644
:hist-forward 281, 624, 644
:hist-viewtree 281, 624, 644
Windows view image symbol
:view-details 280, 624, 644
:view-large-icons 280, 624, 644
:view-list 280, 624, 644
:view-net-connect 281, 624, 644
:view-net-disconnect 281, 624, 644
:view-new-folder 281, 624, 644
:view-parent-folder 281, 624, 644
:view-small-icons 280, 624, 644
windows-menu special slot 124
with-atomic-redisplay macro 659
with-busy-interface macro 660
with-dither macro 806
with-document-pages macro 663
with-external-metafile macro 664
with-geometry macro 667
with-graphics-mask macro 806
with-graphics-post-transla-
tion macro 808
with-graphics-rotation macro 809
with-graphics-scale macro 810
with-graphics-state macro 810
with-graphics-transform macro 812
with-graphics-transform-
reset macro 813
with-graphics-translation macro 814
with-internal-metafile macro 669
with-inverse-graphics macro 815
with-output-to-printer macro 671
without-relative-drawing macro 815
with-page macro 672
with-page-transform macro 673
with-pixmap-graphics-port macro 816
with-print-job macro 673
with-random-typeout macro 674
with-transformed-area macro 817
with-transformed-point macro 818
with-transformed-points macro 819
with-transformed-rect macro 819
:wrap-style initarg 149
wrap-text function 675
wrap-text-for-pane function 676
write-external-image function 820
X
%x% geometry slot 667
:x initarg 166, 405
X window ID 85, 561
X Window System display 81
fallback resources 81
:x-adjust initarg 677
:x-gap initarg 211, 550
:x-ratios initarg 211
:x-uniform-size-p initarg 211
x-y-adjustable-layout class 677
Y
%y% geometry slot 667
:y initarg 166, 405
:y-adjust initarg 677
:y-gap initarg 211
:y-ratios initarg 211
:y-uniform-size-p initarg 211
Z
Z-order
of interfaces 65
of pinboard-objects 270, 402