LispWorks® CAPI Reference Manual

Version 4.1



Copyright and Trademarks

LispWorks CAPI Reference Manual

Version 4.1

October 1998

Part number: 3LADT3A15LF

Copyright © 1994-1998 by Harlequin Group plc.

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

The information in this publication is provided for information only and is subject to change without notice. Harlequin Group plc and its affiliates assume no responsibility or liability for any loss or damage that may arise from the use of any information in this publication. The software described in this book is furnished under license and may only be used or copied in accordance with the terms of that license.

LispWorks is a registered trademark of Harlequin Group plc. Harlequin, Delivery, Transducer/PC, The Authoring Book, ClassWorks, and KnowledgeWorks are all trademarks of Harlequin Group plc.

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

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

CLX and CLUE bear the following copyright notice, which applies to the parts of LispWorks derived therefrom:

Texas Instruments Incorporated, P.O. Box 149149, MS 2151, Austin, Texas 78714-9149

Copyright © 1987, 1988, 1989, 1990, 1991 Texas Instruments Incorporated.

Permission is granted to any individual or institution to use, modify and distribute this software, provided that this complete copyright and permission notice is maintained, intact, in all copies and documentation. Texas Instruments Incorporated provides this software "as is" without express or implied warranty.

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

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

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

US Government Use

The LispWorks Software is a computer software program developed at private expense and is subject to the following Restricted Rights Legend: "Use, duplication, or disclosure by the United States Government is subject to restrictions as set forth in (i) FAR 52.227-14 Alt III or (ii) FAR 52.227-19, as applicable. Use by agencies of the Department of Defense (DOD) is subject to Harlequin's customary commercial license as contained in the accompanying license agreement, in accordance with DFAR 227.7202-1(a). For purposes of the FAR, the Software shall be deemed to be 'unpublished' and licensed with disclosure prohibitions, rights reserved under the copyright laws of the United States. Harlequin Incorporated, One Cambridge Center, Cambridge, Massachusetts 02142."

http://www.harlequin.com/

Europe:

Harlequin Limited Barrington Hall Barrington Cambridge CB2 5RG UK

telephone +44 1223 873 800 fax +44 1223 873 873 support +44 1625 58 8040 North America:

Harlequin Incorporated
One Cambridge Center
Cambridge, MA 02142
USA
talaphana +1 617 374 340

telephone +1 617 374 2400 fax +1 617 252 6505 support +1 617 374 2433 Asia Pacific:

Harlequin Australia Pty. Ltd. Level 12 12 Moore Street Canberra, ACT 2601 Australia

telephone +61 2 6206 5522 fax +61 2 6206 5525 support +44 1625 58 8040

Contents

Preface xiii

1 CAPI Reference Entries 1

abort-dialog 1 activate-pane 2 attach-interface-for-callback 3 beep-pane 3 button 3 button-panel 6 calculate-constraints 9 calculate-layout 10 callbacks 11 call-editor 13 capi-object 13 capi-object-property 14 check-button 15 check-button-panel 16 choice 17 choice-selected-item 19 choice-selected-items 20 collection 21 collection-search 24 collector-pane 25 color-screen 26

column-layout 26

confirm-yes-or-no 28

contain 28

convert-to-screen 29

count-collection-items 30

current-printer 31

define-command 31

define-interface 33

define-layout 36

define-menu 37

destroy 38

display 39

display-dialog 39

display-message 40

display-message-for-pane 40

display-pane 41

drawn-pinboard-object 42

draw-pinboard-object 43

draw-pinboard-object-highlighted 44

 $draw-pinboard-object-unhighlighted\ 44$

editor-pane 44

editor-pane-buffer 46

element 46

ensure-interface-screen 49

exit-confirmer 49

exit-dialog 50

find-string-in-collection 51

form-layout 51

get-collection-item 53

get-constraints 53

get-page-area 54

get-printer-metrics 55

graph-pane 55

grid-layout 58

hide-interface 61

highlight-pinboard-object 61

image-list 62

image-pinboard-object 63

image-set 64

interactive-pane 64

interface 65

interpret-description 69

invalidate-pane-constraints 70

invoke-command 70

invoke-untranslated-command 71

item 71

itemp 73

item-pinboard-object 73

layout 74

line-pinboard-object 75

listener-pane 76

list-panel 76

list-view 77

lower-interface 83

make-container 83

make-general-image-set 84

make-image-locator 85

make-resource-image-set 85

map-collection-items 86

map-pane-children 86

map-typeout 87

menu 88

menu-component 90

menu-item 91

menu-object 92

modify-editor-pane-buffer 94

mono-screen 94

multiple-line-text-input-pane 95

option-pane 95

output-pane 96

over-pinboard-object-p 99

page-setup-dialog 100

pane-adjusted-offset 100

pane-adjusted-position 101

parse-layout-descriptor 102 password-pane 103 pinboard-layout 104 pinboard-object 106 pinboard-object-at-position 107 pinboard-object-overlap-p 108 popup-confirmer 109 print-collection-item 111 print-dialog 112 print-editor-buffer 113 print-file 113 print-text 114 printer-metrics-device-height 114 printer-metrics-device-width 115 printer-metrics-dpi-x 116 printer-metrics-dpi-y 116 printer-metrics-height 117 printer-metrics-width 117 progress-bar 118 prompt-for-confirmation 118 prompt-for-file 119 prompt-for-form 120 prompt-for-forms 122 prompt-for-integer 123 prompt-for-string 124 prompt-for-symbol 125 prompt-with-list 126 push-button 127 push-button-panel 128 quit-interface 129 radio-button 131 radio-button-panel 131 range-pane 132 raise-interface 133 redisplay-interface 133 redisplay-menu-bar 134 redraw-pinboard-layout 135

redraw-pinboard-object 135

remove-capi-object-property 135

right-angle-line-pinboard-object 136

row-layout 137

screen 139

search-for-item 140

set-button-panel-enabled-items 140

set-geometric-hint 141

set-hint-table 141

set-scroll-position 142

set-scroll-range 142

show-interface 142

simple-layout 143

simple-pane 143

simple-pinboard-layout 145

simple-print-port 146

slider 147

switchable-layout 147

tab-layout 148

text-input-choice 151

text-input-pane 151

titled-image-pinboard-object 154

titled-menu-object 155

titled-pane 156

title-pane 158

toolbar 159

toolbar-button 161

toolbar-component 162

toolbar-object 163

top-level-interface 164

top-level-interface-p 165

undefine-menu 165

unhighlight-pinboard-object 165

unmap-typeout 166

update-pinboard-object 166

update-toolbar 166

with-atomic-redisplay 167

with-document-pages 167 with-geometry 168 with-page 169 with-page-transform 169 with-print-job 170 with-random-typeout 170 x-y-adjustable-layout 171

2 GP Reference Entries 173

default-image-translation-table 173 *unit-transform* 174 analyze-external-image 174 apply-rotation 175 apply-scale 175 apply-translation 176 augment-font-description 176 clear-external-image-conversions 177 clear-graphics-port 177 clear-graphics-port-state 178 clear-rectangle 178 compress-external-image 178 compute-char-extents 179 convert-external-image 179 copy-external-image 180 copy-pixels 180 copy-transform 181 create-pixmap-port 182 define-font-alias 183 destroy-pixmap-port 183 dither-color-spec 184 draw-arc 184 draw-arcs 185 draw-character 186 draw-circle 186 draw-ellipse 187 draw-image 188 draw-line 189

draw-lines 189

draw-point 190

draw-points 190

draw-polygon 191

draw-polygons 192

draw-rectangle 193

draw-rectangles 193

draw-string 194

external-image 195

external-image-color-table 195

external-image-color-table 196

externalize-image 196

find-best-font 197

find-matching-fonts 198

font-description 198

font-description-attributes 198

font-description-attribute-value 199

font-fixed-width-p 199

free-image 200

get-bounds 200

get-character-extent 201

get-char-ascent 202

get-char-descent 202

get-char-width 203

get-enclosing-rectangle 203

get-font-ascent 204

get-font-average-width 204

get-font-descent 205

get-font-height 205

get-font-width 205

get-graphics-state 206

get-origin 206

get-string-extent 207

get-transform-scale 208

graphics-port-transform 208

image 208

image-freed-p 209

image-loader 209

image-translation 210

initialize-dithers 210

inset-rectangle 211

inside-rectangle 211

invert-transform 212

load-image 212

make-dither 213

make-font-description 214

make-graphics-state 214

make-image-from-port 215

make-transform 216

merge-font-descriptions 217

offset-rectangle 217

ordered-rectangle-union 217

pixblt 218

pixmap-port 219

port-depth 220

port-height 220

port-string-height 220

port-string-width 221

port-width 221

postmultiply-transforms 222

premultiply-transforms 222

read-and-convert-external-image 223

read-external-image 223

rectangle-bind 223

rectangle-bottom 224

rectangle-height 224

rectangle-left 225

rectangle-right 225

rectangle-top 226

rectangle-union 226

rectangle-width 227

rect-bind 227

 $register\hbox{-}image\hbox{-}load\hbox{-}function~228$

register-image-translation 229

reset-image-translation-table 229 separation 230

set-default-image-load-function 231

set-graphics-port-coordinates 231

set-graphics-state 232

transform 233

transform-area 233

transform-distance 234

transform-distances 234

transform-is-rotated 235

transform-point 235

transform-points 235

transform-rect 236

undefine-font-alias 237

union-rectangle 237

unit-transform-p 238

unless-empty-rect-bind 238

untransform-distance 239

untransform-distances 239

untransform-point 240

untransform-points 240

with-dither 241

with-graphics-mask 241

with-graphics-rotation 242

with-graphics-scale 243

with-graphics-state 243

with-graphics-transform 244

with-graphics-translation 245

with-inverse-graphics 246

without-relative-drawing 246

with-pixmap-graphics-port 247

with-transformed-area 248

with-transformed-point 248

with-transformed-points 249

with-transformed-rect 250

write-external-image 251

Index 253

Preface

This manual contains reference entries for the functions, classes, macros and accessors in the CAPI package and its sub-packages, and the graphics port package. Entries are listed alphabetically, and the typographical conventions used are similar to those used in *Common Lisp: the Language* (2nd Edition) (Steele, 1990). 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 port package is not stricktly part of the CAPI, its reference entires are included in this manual because graphics port functions and macros are usually called from CAPI elements such as output panes. Please also see the relevant chapter in the *CAPI User Guide* for further information on graphics ports.

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", "Syntax", "Superclasses", "Subclasses", "Slots", "Accessors", "Readers", "Compatibility Note", "Description", "Examples", and "See Also".

The default package containing each symbol is the CAPI package in the CAPI reference chapter, and the gp package in the graphics ports reference chapter, unless stated otherwise in the "Package" section of an entry.

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 "Syntax" section provides details of the arguments taken by the functions and macros. Variable arguments for the command are printed *like this*.

Only immediate sub- and superclasses are detailed in the "Subclasses" and "Superclasses" sections of each CAPI class entry.

Examples of the use of commands are given under the "Examples" heading. Some example files can also be found in your installation directory under current-lib/examples/capi/.

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

1

CAPI Reference Entries

The following chapter documents functions exported from the capi package.

abort-dialog Function

Summary The abort-dialog function aborts the current dia-

log.

Syntax abort-dialog &rest ignored-args

Description This function is used to abort the current dialog. For

example, it can made a selection callback from a **Cancel** button so that pressing the button aborts the dialog. In a similar manner the complementary function <code>exit-dialog</code> can be used as a callback for

an OK.

Example (capi:display-dialog

(capi:make-container

:callback 'capi:abort-dialog)
 :title "Test Dialog"))

See also exit-dialog

display-dialog popup-confirmer

interface

activate-pane Function

Summary The activate-pane function gives the focus to a pane and

brings the window containing it to the front.

Syntax activate-pane pane

Description This brings the window containing *pane* to the front, and

gives the focus to the pane (or a sensible alternative inside the same interface if that pane cannot accept the focus).

Example This example demonstrates how to swap the focus from one

window to another.

(setq text-input-pane

(capi:contain (make-instance

'capi:text-input-pane)))

(setq button

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

(capi:activate-pane button)

See also hide-interface

show-interface
quit-interface
simple-pane

attach-interface-for-callback

Function

Summary The attach-interface-for-callback function changes the

interface that is passed when a callback is made.

Syntax attach-interface-for-callback element interface

Description Callbacks for *element* get passed *interface* instead of *element*'s

parent interface.

See also callbacks

element interface

beep-pane Function

Summary The beep-pane function sounds a beep on a screen.

Syntax beep-pane &optional pane

Description This sounds a beep on the screen associated with *pane* or on

the current screen if pane is mil.

Example (capi:beep-pane)

See also simple-pane

screen

button Class

Summary A button is a pane that displays either a piece of text or a

generic image, and that performs an action when pressed. Certain types of buttons can also be selected and deselected.

CAPI Reference Entries

Superclasses simple-pane

item

Subclasses push-button

radio-button check-button

Slots 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 A generic image for the button (or nil).

disabled-image

The image for the button when disabled (or

nil).

enabled If nil the button cannot be selected.

Accessors button-enabled

button-selected

Readers button-image

button-disabled-image

Description The abstract class button is the class that push-button.

radio-button, and check-button are built on. It can be displayed either with text or a generic 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. For more details, see item.

The subclasses of button are just buttons with different interaction styles. It can often be easier just to make an instance of button with the correct interaction (for instance, when the

interaction is only known at run-time). The interaction styles 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 initary interaction and the accessor button-selected.

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

:callback Called when the button is pressed.

:selection-callback

Called when the button is selected.

:retract-callback

Called when the button is deselected.

By default, image and disabled-image are nil, meaning that the button is a text button, but if image is provided then the button displays an image instead of the text. The image can be the name of an image that can be found in the image search path, or an instance of a generic image. The disabled image is the image that is shown when the button is disabled (or nil, meaning that it is left for the window system to decide how to display the image as disabled). For more details about generic images, see the *LispWorks User Guide*.

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.

Example

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

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

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.

Superclasses choice

titled-pane

Subclasses push-button-panel

> radio-button-panel check-button-panel

Slots The type of layout for the buttons. layout-class

> Initialization arguments for the layout. layout-args

> The selection callbacks for each button. callbacks

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 is single selection (meaning that only one of the buttons can be

> selected at any one time), but this can be changed by specify-

ing 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 keyword arguments layout-args is passed to the make-instance.

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

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 1.1 Button and panel classes

Button panel class	Button class
push-button-panel	push-button
radio-button-panel	radio-button
check-button-panel	check-button

For example,

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 :max-width nil in the make-instance.

```
Example
             (capi:contain (make-instance
                             'capi:button-panel
                             :items '(:red :green :blue)
                             :print-function 'string-capitalize))
             (setq buttons (capi:contain
                             (make-instance
                              'capi:button-panel
                              :items '(:red :green :blue)
                              :print-function 'string-capitalize
                              :interaction :multiple-selection)))
             (setf (capi:choice-selected-items buttons)
                     '(:red :green))
             (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)))
See also
             radio-button
             check-button
             push-button
             set-button-panel-enabled-items
```

calculate-constraints

Generic Function

Summary The calculate-constraints generic function calculates the

minimum and maximum size of a pane.

Syntax calculate-constraints pane

Description This generic function calculates the minimum and maximum

size for pane according to the sizes of its children, and sets

these values into pane's geometry cache.

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 chil-

dren.

%min-width% The minimum width of pane.

The maximum width of pane. %max-width%

%min-height% The minimum height of pane.

The maximum height of pane. %max-height%

The constraints of any CAPI element can be found by calling

get-constraints.

See also calculate-layout

> define-layout get-constraints

element layout

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.

Syntax 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 impossi-

ble 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.

Compatibility Note

In LispWorks 3.1, this macro and all of its variable names were in the capi-layouts package. They have been moved

into the CAPI package for simplicity.

See also get-constraints

with-geometry

interpret-description

callbacks Class

Summary The class callbacks is used as a mixin by classes that pro-

vide callbacks.

Superclasses capi-object

Subclasses collection

item

menu-object

Slots callback-type The type of arguments for the callbacks.

selection-callback

The callback for selecting an item.

extend-callback

The callback for extending the selection.

retract-callback

The callback for deselecting an item.

action-callback

The callback for an action.

ACCESSORS callbacks-callback-type

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

Description Each callback function can be one of the following:

Call the function. function Apply the head of the list to the tail. list :redisplay-interface Call redisplay-interface on the top-level interface. :redisplay-menu-bar Call redisplay-menu-bar on the top-level interface. The callback-type specifies which arguments get passed to each of the callbacks. It can take any of the following values, and passes the corresponding data to the callback function: (item-data) :data :data-interface (item-data interface) :interface-data (interface item-data) (item) :item :item-interface (item_interface) :interface-item (interface item) :interface (interface) (item-data item interface) :full :none () nil ()

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* vari-

able means the item itself. The *interface* is the interface of the element.

See also choice

attach-interface-for-callback

call-editor Generic Function

Summary The call-editor generic function executes an editor com-

mand in an editor pane.

Syntax call-editor editor-pane command

Description This executes the editor command command in the current

buffer in editor-pane.

Example (setq editor (capi:contain

(make-instance 'capi:editor-pane

:text "abc")))

(capi:call-editor editor "End Of Buffer")

See also editor-pane

capi-object Class

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

Superclasses standard-class

Subclasses item

callbacks element interface

Slots name The name of the object.

A property list for storing miscellaneous

information.

ACCESSORS capi-object-name

capi-object-plist

Description The class capi-object provides a name and a property list

for general purposes, along with the accessors

capi-object-name and capi-object-plist respectively. A capi-object's name is defaulted by define-interface to be

the name of the slot into which the object is put.

Examples (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 set properties

in a property list.

Syntax capi-object-property object property

Description All CAPI objects contain a property list, similar to the symbol

plist. The recommended ways of setting properties are capi-object-property and (setf capi-object-property).

To remove a property, use the function

remove-capi-object-property.

Example In this example a list panel is created, and a test property is

set and examined using capi-object-property.

check-button Class

Summary A check button is a button that can be either selected or dese-

lected, and its selection is independent of the selections of

any other buttons.

Superclasses button

titled-pane

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.

The button can be selected and deselected using this code.

(setf (capi:button-selected button) t)

```
(setf (capi:button-selected button) nil)
               The following code disables and enables the button.
               (setf (capi:button-enabled button) nil)
               (setf (capi:button-enabled button) t)
 See also
               push-button
               radio-button
               button-panel
check-button-panel
                                                                    Class
 Summary
               A check-button-panel is a pane containing a group of but-
               tons each of which can be selected or deselected.
 Superclasses
               button-panel
 Description
               The class check-button-panel inherits all of its behavior
               from button-panel, which itself inherits most of its behavior
               from choice. Thus, the check-button-panel can accept
               items, callbacks, and so on.
 Example
               (capi:contain (make-instance
                               'capi:check-button-panel
                               :title "Select some packages"
                               :items '("CAPI" "LISPWORKS" "CL-USER")))
                (setq buttons (capi:contain
                               (make-instance
                                'capi:check-button-panel
                                :title "Select some packages"
                                :items '("CAPI" "LISPWORKS" "CL-USER")
                                :layout-class 'capi:column-layout)))
                (capi:choice-selected-items buttons)
 See also
               check-button
               push-button-panel
```

radio-button-panel

choice Class

Summary A choice is an abstract class that collects together a group of

items, and provides functionality for displaying and select-

ing them.

Superclasses: collection

Subclasses list-panel

button-panel option-pane graph-pane menu-component

Slots 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

It t, retains any selection when the items

change.

ACCESSORS choice-selection

choice-selected-item
choice-selected-items

Readers choice-interaction

Description The class choice inherits most of its behavior from collec-

tion, and then provides the selection facilities itself. The classes list-panel, button-panel, 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 interactions, and these control how it behaves when an item is selected by the user.

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

:single-selection

The choice can have only one selected item.

:multiple-selection

The choice can have multiple selected items.

:extended-selection

An alternative to multiple-selection.

In no-selection mode, the choice cannot have a selection, and so behaves just as a collection would.

In single-selection mode, the choice can only have one item selected at a time. When a new selection is made, the old selection is cleared and the selection-callback is called.

In multiple-selection mode, 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.

In extended-selection mode, the choice can have any number of items selected as with multiple-selection mode, 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.

The choice's selection stores the indices of the currently selected item, and is a single number for single selection choices and a list for all other interactions. The complementary accessors 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 set to t, then the selection is preserved over the change.

Example

The following example defines a choice with three possible selections.

See also

choice-selected-item

choice-selected-item

Generic Function

Summary The function choice-selected-item returns the currently

selected item in a single selection choice.

Syntax choice-selected-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. It is an error to

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

Example

First we set up a single selection choice — in this case, a list panel.

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.

```
(setf (capi:choice-selected-item list) 'e)
(capi:choice-selected-item list)
```

See also

choice

choice-selected-items

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.

Syntax

choice-selected-items choice

Description

The function choice-selected-items returns the currently selected items in a choice as a list of the items. A setf method is provided as a means of setting the currently selected items. 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.

Compatibility Note In LispWorks 3.1, choice-selected-items returned a single item for single-selection list panels but this was considered to cause problems and so it was changed to always return a list, and the alternative function

choice-selected-item was provided for single selection

panes.

Examples

First we set up a multiple selection choice — in this case, a list panel.

The following code line returns the selections of the list.

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

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

```
(setf (capi:choice-selected-items list) '(a c e))
(capi:choice-selected-items list)
```

See also choice

choice-selected-item

collection Class

Summary A collection collects together a set of items, and provides

functionality for accessing and displaying them.

Superclasses capi-object

callbacks

Subclasses choice

Slots 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.

item-map-function

A function that maps a function over the items.

ACCESSORS collection-items

collection-print-function collection-test-function

Readers collection-items-count-function

collection-items-get-function collection-items-map-function

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 can either be arbitrary Common Lisp objects which can be printed with the print-function, or can be instances of the CAPI class item in which case they are displayed with the text field of the item. 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 they are specified.

By default, the items must be a sequence, but this can be changed by specifying an items-get-function, an items-count-function, and an item-map-function. The items-get-function should expect the items and an index, and should return the indexed item. The items-count-function should expect the items and should return the number of them. The item-map-function should expect the items, a function and a flag collect-results-p, and should call the function on each of the items in return. If collect-results-p is non-nil, then it should also collect the results of these calls together in a list.

Examples

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

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

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 intereface)
               (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)
                             :min-width 200 :min-height 200)
See also
             item
             count-collection-items
             get-collection-item
             map-collection-items
             search-for-item
```

collection-search

Generic Function

Summary	The generic function collection-search calls find-string-in-collection with a string provided by the user.
Syntax	collection-search collection &optional set
Description	Prompt the user for a string and call find-string-in-collection with this string.
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.

Superclasses editor-pane

Slots The name of a buffer onto an editor stream. buffer-name

> The editor stream to be collected. stream

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 a buffer-name of a non-existent stream or specify nothing, in which case the CAPI will create a unique buffer name for you. To then get hold of that stream, use the reader collector-pane-stream on the collector-pane.

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

Examples

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

```
(setq collector (capi:contain
                 (make-instance 'capi:collector-pane)))
(setq *test-stream*
  (capi:collector-pane-stream collector))
(capi:contain
  (make-instance 'capi:collector-pane
                 :stream *test-stream*))
(format *test-stream* "Hello World~%")
```

Finally, this example shows how to create a collector pane

onto the "Background Output" stream.

See also with-random-typeout

map-typeout
unmap-typeout

color-screen Class

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 mono-screen

column-layout Class

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

Superclasses grid-layout

Slots ratios The size ratios between the layout's chil-

dren.

adjust The horizontal adjustment for each child.

The gap between each child.

uniform-size-p

If ${\tt t}$, 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 keywords y-ratios, x-adjust, y-gap and y-uniform-size-p. Example (capi:contain (make-instance 'capi:column-layout :description (list (make-instance 'capi:push-button :text "Press me") "Title" (make-instance 'capi:list-panel :items '(1 2 3))))) (setg 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))) (setf (capi:layout-x-adjust column) :right) (setf (capi:layout-x-adjust column) :left) (setf (capi:layout-x-adjust column) :center)

See also

row-layout

confirm-yes-or-no

Function

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

containing a message and a Yes and No button.

Syntax confirm-yes-or-no format-string &rest format-args

Description This pops up a dialog box containing a message and the but-

tons **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 Com-

mon 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

(when (capi:confirm-yes-or-no "Close ~S?" pane)

(capi:quit-interface pane))

See also prompt-for-confirmation

display-dialog popup-confirmer

contain Function

Summary The contain function creates a container for a CAPI element

and is provided as a convenient way of testing CAPI func-

tionality.

Syntax contain element &key screen &allow-other-keys

Description

This displays a window containing *element* and returns *element* as its result. It is mainly used during interactive sessions as a convenient way of testing CAPI functionality, and many of the CAPI examples use it for this purpose. The container is created using make-container, which can make containers for any of the following classes:

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.

```
Example
```

```
(capi:contain (make-instance 'capi:text-input-pane))
             (capi:contain (make-instance
                             'capi:column-layout
                             :description '("Title:"
                                            ,(make-instance
                                            'capi:text-input-pane))))
             (capi:contain (make-instance 'capi:menu-item
                                           :title "Test"))
See also
             make-container
             display
```

convert-to-screen

element

Function

Summary

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

Syntax convert-to-screen &optional object

Description This finds the appropriate screen for the CAPI object object. If

object is nil it returns the default screen, and if it is a number

n it returns the *n*th screen.

Examples (capi:convert-to-screen)

Assuming you have a second screen, you can access it using:

(capi:convert-to-screen 1)

See also screen

count-collection-items

Generic Function

Summary The count-collection-items generic function returns the

number of items in a collection.

Package capi-internals

Syntax count-collection-items collection &optional representation

Description Return the number of items in the collection by calling the

items-count-function. If representation is non-nil, it is used in place of the items-representation in the choice.

Examples The following example uses count-collection-items to

return the number of items in a list panel.

(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-printer

Function

Summary Returns the currently selected printer object for the specified

library.

Syntax current-printer &optional library interactive

Description The current-printer function returns the currently selected

printer object for the library specified by *library*. If *library* is nil, the default library is used. If *interactive* is non-nil and there is no current printer a confirmer is displayed warning

the user.

See also page-setup-dialog

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.

Syntax 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.

Examples

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

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

```
(capi:define-command
  :select-object (:button-1 :press)
  :translator #'(lambda (output-pane x y)
                  (let ((object
                      (capi:pinboard-object-at-position
                                      output-pane x y)))
                                   (when object
                                     (list object)))))
(defun object-select-callback (output-pane
                               &optional object)
  (when object (capi:display-message
                "Pressed on ~S in ~S"
                object output-pane)))
(setq pinboard
      (capi:contain (make-instance
                    'capi:pinboard-layout
                    :input-model '((:select-object
                            object-select-callback)))))
(make-instance 'capi:item-pinboard-object
               :text "Press Me!"
               :parent pinboard
               :x 10 :y 20)
```

(make-instance 'capi:line-pinboard-object

:parent pinboard
:x 20 :y 50

:width 100 :height 100)

See also output-pane

invoke-command

invoke-untranslated-command

define-interface Macro

Summary The define-interface macro defines subclasses of

interface.

Syntax 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 macro is essentially a version of defclass which accepts

the following extra options:

:panes Descriptions of the interface's panes.

:layouts Descriptions of the interface's layouts.

menus Descriptions of the interface's menus.

:menu-bar A list of menus for the interface's menu bar.

:definition Options to alter define-interface.

The options :panes, :layouts and :menus add extra slots to the class that will contain the CAPI object described in their description. Within the scope of the extra options, the slots themselves are available by referencing the name of the slot, and the interface itself is available with the variable capi:interface. Each of the slots can be made to have readers, writers or accessors by passing the appropriate defclass keyword as one of the optional arguments in the description.

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

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

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

The :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 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)
```

where *slot-name* is the slot name for each menu or menu component, *title* is the menu's title or the keyword :component, *descriptions* is a list of menu item descriptions, and *initargs* is a list of the initialization arguments for the menu. Each item

description is either a title, or a list of items containing a title and a list of initialization arguments for the item.

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.

Compatibility Note

The variable that was bound to the interface itself was called capi::self in LispWorks 3.1, and has been kept for compatibility. However, it is recommended that you use capi:interface or specify your own name using :interface-variable.

Examples

Firstly, a couple of pane examples:

```
(capi:define-interface test1 ()
   ()
   (:panes
        (text capi:text-input-pane))
   (:default-initargs :title "Test1"))
(capi:display (make-instance 'test1))
(capi:define-interface test2 ()
   ()
   (:panes
        (text capi:text-input-pane)
        (buttons capi:button-panel :items '(1 2 3)))
   (:layouts
        (main-layout capi:column-layout '(text buttons)))
   (:default-initargs :title "Test2"))
(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
                 (color-menu "Colors" ((:component
                                           (:red :green :blue)
                                       :interaction :single-selection
                                         :print-function
                                          'string-capitalize))))
               (:menu-bar color-menu)
               (:default-initargs :title "Test4"))
             (capi:display (make-instance 'test4))
See also
             interface
             layout
             menu
```

define-layout Macro

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

Syntax 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.

Compatibility Note In LispWorks 3.1, this macro and all of its variable names were in the capi-layouts package. They have been moved into the CAPI package for simplicity.

See also

interpret-description
calculate-constraints
calculate-layout
layout

define-menu Macro

Summary The define-menu macro defines a menu function.

Syntax define-menu function-name (self) title menu-body &rest menu-

options

Description

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

Example (capi:define-menu make-test-menu (self)

```
"Test"
("Item1"
"Item2"
"Item3"
(:component
("Item4"
"Item5")
:interaction :single-selection)))
(setq interface (make-instance 'capi:interface))
```

destroy Generic Function

Summary The generic function destroy closes the window associated

with an interface and then calls the interface's

destroy-callback.

Syntax destroy interface

Description This 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.

```
Example (setq interface
```

(capi:destroy interface)

See also interface

quit-interface

display Function

Summary The display function displays a CAPI interface on a speci-

fied screen.

Syntax display interface &key screen

Description The function display displays the CAPI interface interface on

> the specified *screen* (or the current one if not supplied). Use the function contain to display objects other than interfaces.

Example (capi:display (make-instance 'capi:interface

:title "Test"))

See also interface

contain

display-dialog quit-interface

Function display-dialog

Summary The display-dialog function displays a CAPI interface as a

dialog box.

Syntax display-dialog interface &key screen focus (modal t)

Description This is a complementary function that displays the CAPI

interface interface as a dialog box.

The variable *screen* is the screen for the dialog to be displayed on. The *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. The variable *modal* indicates whether or not the dialog takes over all input to the

application.

The CAPI also provides popup-confirmer which gives you

the standard OK and Cancel button functionality.

EXample (capi:display-dialog (capi:make-container

(make-instance

'capi:push-button-panel
:items '("OK" "Cancel")
:callback-type :data

:title "Empty Dialog"))

See also interface

popup-confirmer
exit-dialog
abort-dialog
display

display-message

Function

Summary The function display-message displays a message on the

current CAPI screen.

Syntax 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.

Example (capi:display-message "Current screen = ~S"

(capi:convert-to-screen))

See also 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.

Syntax 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.

Example (setq pane (capi:contain (make-instance

'capi:text-input-pane)))

(capi:display-message-for-pane pane

"Just created ~S" pane)

See also display-message

display-pane Class

Summary The class display-pane is a pane that displays several lines

of text.

Superclasses titled-pane

Slots 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.

```
Examples
              (capi:contain (make-instance
                             'capi:display-pane
                               '("One" "Line" "At" "A" "Time...")))
              (setq display-pane (capi:contain
                                   (make-instance
                                     'capi:display-pane
                                     :text
                                    '("One" "Line" "At" "A" "Time...")
                                     :min-height '(:character 5))))
             (setf (capi:display-pane-text display-pane)
                    '("Some" "New" "Text"))
See also
             title-pane
             text-input-pane
             editor-pane
```

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.

Superclasses pinboard-object

Slots display-callback

Called to display the object.

ACCESSORS drawn-pinboard-object-display-callback

Description The display-callback is called with the output pane to

draw on, the drawn-pinboard-object itself, and the x, y, width and height of the object, and it is expected to redraw

that section.

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

Example

See also pinboard-layout

draw-pinboard-object

Generic Function

Syntax

draw-pinboard-object pinboard object

&key x y width height &allow-other-keys

Description

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

Compatibility

Note that draw-pinboard-object now takes keyword arguments, while in LispWorks 3.1 it did not.

See also

pinboard-layout
pinboard-object

draw-pinboard-object-highlighted

Generic Function

Summary The generic function draw-pinboard-object-hilighted

draws highlighting on a pre-drawn pinboard object.

Syntax draw-pinboard-object-highlighted pinboard object

&key &allow-other-keys

Description This generic function draws the highlighting onto a pinboard

object that has already been drawn. The default highlighting method draws a box around the object, and should be suffi-

cient for most purposes.

See also draw-pinboard-object-unhighlighted

highlight-pinboard-object

draw-pinboard-object-unhighlighted

Generic Function

Summary The generic function draw-pinboard-object-unhighlighted

removes the highlighting from a pinboard object.

Syntax draw-pinboard-object-unhighlighted pinboard object

&key &allow-other-keys

Description This generic function removes the highlighting from a pin-

board object.

See also draw-pinboard-object-highlighted

highlight-pinboard-object

editor-pane Class

Summary An editor pane is an EMACS-style editor that has all of the

functionality described in the LispWorks Guide To The Editor.

Superclasses output-pane

Subclasses interactive-pane

collector-pane

Slots text The text in the editor pane.

enabled If t the editor pane will accept input from

the mouse and keyboard.

buffer-name The name of the editor buffer.

ACCESSORS editor-pane-text

editor-pane-enabled

Description The accessor editor-pane-text is provided to read and

write the text in the editor buffer. The accessor

editor-pane-enabled is used to enable and disable the editor (when it is disabled, it ignores all input from the mouse

and keyboard).

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

the editor-pane uses some existing buffer.

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

(setq editor (capi:contain

:enabled hill)))

(setf (capi:editor-pane-enabled editor) t)

See also call-editor

modify-editor-pane-buffer

editor-pane-buffer

Accessor Function

Summary The editor-pane-buffer function returns the editor buffer

associated with an editor pane.

Syntax editor-pane-buffer pane

Description This accessor function returns the editor buffer associated

with an editor pane, which can be manipulated in the stan-

dard ways with the routines in the editor package.

Example (setq editor-pane

(setq buffer

(capi:editor-pane-buffer editor-pane))

(editor:buffer-insert-string buffer

"Some more text...")

See also editor-pane

element Class

Summary The class element is the superclass of all CAPI objects that

appear in a window.

Superclasses capi-object

Subclasses simple-pane

pinboard-object

menu

collection

Slots parent The element containing this element.

interface The interface containing this element.

min-width The minimum width of the element.

min-height The minimum height of the element.

max-width The maximum width of the element.

max-height The maximum height of the element.

 \mathbf{x} The x position for the element in a pinboard.

The y position for the element in a pinboard.

ACCESSORS element-parent

Readers element-interface

Description

The class element contains the slots parent and interface which contain the element and the interface that the element is contained in respectively. The writer method element-parent can be used to re-parent an element into another parent (or to remove it from a container entirely by setting its parent to nil).

All elements accept hints as to the initial size and position of the element in question. By default elements have a minimum pixel size of one by one, and a maximum size of nil (meaning no maximum), but any of the following hints can be specified to change these values:

The x position of the element.

The y position of the element.

:min-width The width of the element.

:min-height The minimum height of the element.

:max-width The maximum width of the element.

:max-height The maximum height of the element.

:best-width The initial width of the element.

:best-height The initial height of the element.

The possible values for these hints are as follows:

integer The size in pixels.

t For max-width and max-height, means use

the minimum.

:text-width The width of any text in the element.

:screen-width The width of the screen.

:screen-height The height of the screen.

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

The maximum size of the hints.

The minimum size of the hints.

+ The sum of the hints.

- The subtraction of hints from the first.

* The multiplication of the hints.

/ The division of hints from the first.

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.

Examples

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

(setq object (make-instance

'capi:image-pinboard-object

:x 10 :y 10

:image "new-lispworks-logo"

:parent pinboard))

(setf (capi:element-parent object) nil)

(setf (capi:element-parent object) pinboard)

See also set-hint-table

ensure-interface-screen

Function

Summary The ensure-interface-screen function ensures that a top

level interface is displayed on a given screen.

Syntax ensure-interface-screen self &key screen

Description This ensures that the top level interface is displayed on the

given screen (or the default) if display is called later without a :screen argument. This allows the querying of font and color information associated with a particular screen. It

returns the screen that is used.

See also screen

display interface

exit-confirmer Function

Summary The exit-confirmer function is called by the **OK** button on a

dialog created with popup-confirmer.

Syntax exit-confirmer &rest dummy-args

Description This is the function that is called by the **OK** button on a dialog

created using popup-confirmer, and it is provided as an entry point so that other callbacks can behave in the same way. There is a full description of the **OK** button in popup-con-

firmer.

Example This example demonstrates the use of exit-confirmer to

make the dialog exit when pressing return in the text input pane. It also demonstrates the use of value-function as a means of deciding the return value from popup-confirmer.

(capi:popup-confirmer (make-instance

'capi:text-input-pane

:callback 'capi:exit-confirmer)

"Enter some text:"
:value-function

'capi:text-input-pane-text)

See also popup-confirmer

display-dialog

interface

exit-dialog Function

Summary The exit-dialog function exits the current dialog.

Syntax exit-dialog value

Description This function is the means to successfully return a value from

the current dialog. Hence, it might be called from an **OK** button so that pressing the button would cause the dialog to return successfully, whilst the **Cancel** button would call the

counterpart function abort-dialog.

Example (capi:display-dialog

(capi:make-container

(make-instance 'capi:text-input-pane

:callback-type :data

:callback 'capi:exit-dialog)

:title "Test Dialog"))

See also abort-dialog

display-dialog popup-confirmer

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.

Syntax 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

form-layout Class

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

Superclasses layout

Slots The gap between rows in the form. vertical-gap

vertical-adjust

The adjustment made to the rows.

The gap between the two columns. title-gap

The adjustment made to the left column. title-adjust

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

Note

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 alterna-

tive to forms.

Examples (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))))

(capi:contain (make-instance

'capi:grid-layout :description children :x-adjust '(:right :left) :y-adjust :center))

See also grid-layout

layout

get-collection-item

Generic Function

Summary The generic function get-collection-item returns the item

at a specified position in a collection.

Syntax get-collection-item self index

Description The generic function get-collection-item returns the item

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

index for a given item in a collection.

See also collection

search-for-item

get-constraints Function

Summary The get-constraints function returns a list of the con-

straints for an element.

Package capi-internals

Syntax get-constraints element

Description The function get-constraints returns the constraints for ele-

 $\it ment$ 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 calculateconstraints to calculate these sizes initially, but then just uses the values in the geometry cache for the element. To force an element to take account of its new constraints, call

the function invalidate-pane-constraints.

See also calculate-constraints

define-layout

element

invalidate-pane-constraints

get-page-area Function

Summary Calculates the dimensions of suitable rectangles for use with

with-page-transform.

Syntax get-page-area printer &key scale dpi screen

Description The get-page-area function is provided to simplify the cal-

culation 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 spec-

ified.

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-device-height

printer-metrics-device-width

printer-metrics-dpi-x
printer-metrics-dpi-y

printer-metrics-height
with-page-transform

get-printer-metrics

Function

Summary Returns a printer-metrics object for a printer.

Syntax get-printer-metrics printer

Description The get-printer-metrics functions takes a printer as its

argument and returns a printer-metrics object.

See also printer-metrics-device-height

printer-metrics-device-width

printer-metrics-dpi-x
printer-metrics-dpi-y
printer-metrics-height
with-page-transform

graph-pane Class

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

graph.

Superclasses simple-pinboard-layout

choice

x-y-adjustable-layout

Slots roots The roots of the graph.

children-function

Returns the children of a node.

layout-function

A function to layout the nodes.

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-roots

Description

A graph pane calculates the graph by calling the children-function on each of its roots, and then calling it again on each of the children recursively until it runs out of children. The children-function gets called with a node of the graph and should return a list of the children of that node.

The layout-function tells the graph pane how to lay out its children. It can take two values:

:left-right Lay the graph out from the left to the right.

:top-down Lay the graph out from the top down.

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 node for each pane, and by default it creates an instance of node-pinboard-class. It gets passed the graph pane and the node, and should return an instance of simple-pane or pinboard-object.

```
Examples
             (defun node-children (node)
               (when (< node 16)
                      (list (* node 2)
                            (1+ (* node 2)))))
              (setq graph (capi:contain
                            (make-instance 'capi:graph-pane
                                           :roots '(1)
                                           :children-function
                                              'node-children)
                                           :best-width 300
                                           :best-height 400))
             (setf (capi:graph-pane-roots graph) '(2 6))
              (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
                                           :x-adjust :left)
```

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

:best-width 300
:best-height 400)

This example demonstrates the use of :node-pinboard-class to specify that the nodes are drawn as push buttons.

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

grid-layout Class

Summary The grid-layout is a layout which positions its children on a

two dimensional grid.

Superclasses x-y-adjustable-layout

Subclasses row-layout column-layout

Slots orientation The orientation of the children.

rows The number of rows in the grid.

columns The number of columns 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 min-width of any column is the maximum of the min-widths of the children in the column. The size of the layout is controlled by the constraints on the rows and columns.

The description is either a two dimensional array or a list in the order specified by the orientation (which defaults to :row). In the latter case, one of :columns or :rows can be given to specify the dimensions (the default is two columns).

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

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

(round (* total ratio) ratio-sum)

where ratio-sum is the sum of the non-nil elements of x-ratios (or y-ratios) and ratio is the element of ratios corresponding to the child. If this ideal ratio size does not fit the maximum or minimum constraints on the child size, and the constraint means that changing the ratio size would not assist the sum of the child sizes fitting the total space available, then the child is fixed at its constrained size, the child is removed from the ratio calculation, and the calculation is performed again. If x-ratios (or y-ratios) has fewer elements than the number of children, 1 is used for each of the

missing ratios. Leaving x-ratios (or y-ratios) nil causes all of the children to be the same size.

The positions of each pane in the layout can be specified using x-adjust and y-adjust like every other x-y-adjustable-layout, except that if there is one value then it is used for all of the panes, whereas if it is a list then each value in the list refers to one row or column. If the list does not contain a value for every row or column then the last value is taken to refer to all of the remaining panes.

```
Example
              (capi:contain (make-instance
                             'capi:grid-layout
                             :description '("1" "2" "3"
                                             "4" "5" "6"
                                             "7" "8" "9")
                             :columns 3))
              (capi:contain (make-instance
                             'capi:grid-layout
                             :description (list "List:"
                                                 (make-instance
                                                  'capi:list-panel
                                                  :items '(1 2 3))
                                                  "Buttons:"
                                                 (make-instance
                                                  'capi:button-panel
                                                  :items '(1 2 3)))))
              (capi:contain (make-instance
                             'capi:grid-layout
                             :description (list "List:"
                                                 (make-instance
                                                  'capi:list-panel
                                                  :items '(1 2 3))
                                                 "Buttons:"
                                                 (make-instance
                                                  'capi:button-panel
                                                  :items '(1 2 3)))
                             :x-adjust '(:right :left)
                             :y-adjust '(:center :bottom)))
```

See also layout

hide-interface Function

Summary The function hide-interface hides the interface containing

a specified pane.

Syntax hide-interface pane &optional iconify

Description The function hide-interface hides the interface containing

pane from the screen. If *iconify* is non-mil then it will iconify it, else it will just remove it from the screen. To show it again,

use show-interface.

See also interface

show-interface quit-interface

highlight-pinboard-object

Generic Function

Summary The highlight-pinboard-object generic function highlights

a specified pinboard object.

Syntax highlight-pinboard-object pinboard object &key (redisplay t)

Description This function causes the pinboard object *object* to become

highlighted until unhighlight-pinboard-object is called on it. If redisplay is non-nil, then the pinboard object highlight-

ing is drawn immediately using

draw-pinboard-object-highlighted.

Compatibility

In LispWorks 3.1, this function only drew the highlight. That

Note functionality is performed by

draw-pinboard-object-highlighted.

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.

Superclasses None

Slots 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 initary specifies a list. Each item in the list

may be one of the following.

A pathname or string

This specifies the filename of a file suitable for loading with gp:load-image. Currently

this must be a BMP file.

A symbol The symbol must have been previously reg-

istered by means of a call to gp:register-

image-translation.

An image object, as returned by gp: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.

image-pinboard-object

Class

Summary An image pinboard object is a pinboard object that displays

itself as a generic image.

Superclasses pinboard-object

Subclasses titled-image-pinboard-object

Slots image The generic image to be displayed.

Accessors image-pinboard-object-image

Description The image argument to image-pinboard-object should

either be a generic image itself, or the name of a file containing a generic image that is in a directory in your image search

path. The image displayed in the object can be changed

dynamically using the writer function

(setf image-pinboard-object-image)

Compatibility The class image-pinboard-object was called

Note icon-pinboard-object in LispWorks 3.1. The class

icon-pinboard-object is provided for compatibility, but

may be removed in the future.

Example (setf image (capi:contain

(make-instance

'capi:image-pinboard-object
:image "new-lispworks-logo")))

(setf (capi:image-pinboard-object-image image)

"diamond1")

(setf (capi:image-pinboard-object-image image)

"new-lispworks-logo")

See also pinboard-layout

image-set 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:

See also make-general-image-set

make-resource-image-set

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 func-

tionality for handling Lisp forms.

Superclasses editor-pane

Subclasses listener-pane

Slots stream The stream used for I/O.

top-level-function

The input processing function.

Readers interactive-pane-stream

interactive-pane-top-level-function

Description The top-level-function is called to process any input that

comes into the stream. The first argument is the interface containing the interactive pane. The second argument is the interactive pane itself. The third argument is the Common Lisp I/O stream. The function should read from the stream to activate the interactive-pane. The default runs a Lisp listener

top-loop.

Compatibility 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 have been kept for compatibility but may be dropped in future versions of Lisp-

Works.

See also collector-pane

Note

interface Class

Summary The class interface is the top level window class, which

contains both menus and a hierarchy of panes and layouts. Interfaces can also themselves be contained within a layout,

in which case they appear without their menu bar.

Superclasses simple-pane

titled-pane

Slots title The title of the interface.

layout The layout of the interface.

menu-bar-items

The items on the menu bar.

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.

ACCESSORS interface-title

pane-layout

interface-menu-bar-items
interface-destroy-callback

interface-confirm-destroy-function

Description Every interface can have a 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-pane 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. Note that an interface may have some automatic menus created by the environment in which it is running (for example the **Works** menu in the LispWorks environment). To switch these automatic menus off, specify the argument

: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. Then to remove it from the display again, 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.

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 max-width and max-height for elements.

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

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

```
(capi:display (make-instance
                              'capi:interface
                              :title "Test Interface"
                              :confirm-destroy-function
                              #'(lambda (interface)
                                        (capi:confirm-yes-or-no
                                         "Really quit ~S"
                                         interface))))
             (capi:display (make-instance
                              'capi:interface
                              :menu-bar-items
                                (list
                                  (make-instance 'capi:menu
                                                  :title "Menu"
                                                  :items '(1 2 3)))
                              :title "Menu Test"))
             (setq interface (capi:display
                                (make-instance
                                  'capi:interface
                                  :title "Test Interface"
                                  :layout (make-instance
                                             'capi:simple-layout
                                             :description
                                               (make-instance
                                                'capi:text-input-pane)
                                           ))))
             (setf (capi:pane-layout interface)
                    (make-instance 'capi:simple-layout
                                    :description
                                  (make-instance 'capi:editor-pane)))
             (capi:display (make-instance
                             'capi:interface
                             :title "Test"
                             :best-x 200
                             :best-y 200
                             :best-width '(/ :screen-width 2)
                             :best-height 300))
See also
             layout
             switchable-layout
             menu
             display
             display-dialog
```

quit-interface
define-interface
activate-pane

interpret-description

Generic Function

Summary The generic

The generic function interpret-description converts an abstract description of a layout's children into a list of the

children's geometry objects.

Syntax interpret-description layout description interface

Description The generic function interpret-description translates an

abstract description of the *layout*'s children into a list of those

children's geometry objects.

For example, 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

parse-layout-descriptor to do the translation for each

item.

Compatibility
Note

In LispWorks 3.1, this macro and all of its variable names were in the capi-layouts package. They have been moved

into the CAPI package for simplicity.

See also parse-layout-descriptor

define-layout

layout interface

invalidate-pane-constraints

Function

Summary The invalidate-pane-constraints function is used to cause

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

Syntax 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 The invoke-command function invokes a command in the

input model for a specified output pane.

Syntax 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

 ${\tt invoke-untranslated-command}.$

See also invoke-untranslated-command

define-command
output-pane

invoke-untranslated-command

Function

Summary The invoke-untranslated-command function invokes a com-

mand in the input model for a specified output pane, but without the translator being called to process gesture infor-

mation.

Syntax invoke-untranslated-command command output-pane &rest

event-args

Description This 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.

Superclasses callbacks

capi-object

Subclasses menu-item

button

item-pinboard-object

Slots data The data associated with the item.

text The text to appear in the item (or nil).

print-function

If no text, this is called to print the data.

selected If t the item is selected.

Accessors 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 displayed as a string using its text if specified, or else by calling 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.

(defun main-callback (data interface)

```
Example
```

```
(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

itemp Generic Function

Syntax itemp object

Description This is equivalent to

(typep object 'capi:item)

See also item

collection

item-pinboard-object

Class

Summary An item-pinboard-object is a pinboard-object that dis-

plays a single piece of text.

Superclasses pinboard-object

item

Subclasses titled-image-pinboard-object

Slots font The font to draw the item in (or nil).

foreground The foreground color (or nil).

Accessors item-pinboard-object-font

item-pinboard-object-foreground

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

layout Class

Summary A layout is a simple pane that positions one or more child

panes within itself according to a layout policy.

Superclasses titled-pane

Subclasses simple-layout

grid-layout pinboard-layout switchable-layout

Slots default A flag to mark the default layout for an

interface.

description The list of the layout's children.

ACCESSORS layout-description

Description The layout description is an abstract description of the chil-

dren of the layout, and each layout defines its format. Generally, the description is a list of either panes, slot names (where the name refers to a slot in the layout's interface containing a pane) or strings (where the string gets converted to a titlepane). 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.

See also define-layout

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.

Superclasses pinboard-object

Subclasses right-angle-line-pinboard-object

Slots direction The direction of the line (:up or :down).

Description If the direction is :down, then the line is drawn from top-left

to bottom-right, and if it is :up it is drawn from bottom-left to

top-right. 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

:min-width 100
:min-height 100))

(capi:contain (make-instance

'capi:line-pinboard-object

:min-width 100
:min-height 100
:direction :up))

See also pinboard-layout

listener-pane Class

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 dis-

played.

Example (capi:contain (make-instance 'capi:listener-pane)

:best-width 300 :best-height 200)

See also collector-pane

interactive-pane

list-panel Class

Summary The class list-panel is a pane that can display a group of

items and provides support for selecting items and perform-

ing actions on them.

Superclasses choice

titled-pane

Subclasses list-view

Description The class list-panel gains most of its behavior from choice,

which is an abstract class that handles items and their selec-

tion. 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.

```
Example
             (setq list (capi:contain
                           (make-instance 'capi:list-panel
                                          :items '(:red :blue :green)
                                          :print-function
                                             'string-capitalize)))
             (setf (capi:choice-selected-item list) :red)
              (setf (capi:choice-selected-item list) :green)
             (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))))
```

list-view Class

Summary The list view pane is a choice that displays its items as icons

and text in a number of formats.

Superclasses list-panel

See also

Slots view Specifies which view the list view pane

shows. The default is :icon.

subitem-function

button-panel

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.

columns Defines the columns used in report view

auto-reset-column-widths

Determines whether columns automatically resize. Defaults to :all.

use-large-images

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

use-small-images

Indicates whether small icons will be used. Defaults to t.

use-state-images

Indicates whether state images will be used. Defaults to nil.

large-image-width

Width of a large image. Defaults to 32.

large-image-height

Height of a large image. Defaults to 32.

small-image-width

Width of a small image. Defaults to 16.

small-image-height

Height of a small image. Defaults to 16.

state-image-width

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

state-image-height

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

Accessors

list-view-view

list-view-subitem-function

list-view-subitem-print-functions

list-view-image-function

list-view-state-image-function

list-view-columns

list-view-auto-reset-column-widths

Description

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

The list view displays its items in one of four ways, initially determined by the :view initarg, and subsequently changed by (setf list-view-view). An application may use the list view pane in just a single view, or may change the view between all four available views.

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

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

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

In this view, small icons are displayed, together with their label, positioned in the space available.

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

• The report view — :report

:align

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

The column heading

The width of the column in pixels. If this keyword is omitted or has the value mil, 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.

The sub-item-function is called on the item to return subitem objects that represent the additional information to be displayed in the subsequent columns. Hence, the subitemfunction 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. The subitem-print-functions 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 gp:load-image. Currently this must be a bitmap file.

A symbol

The symbol must have been previously registered by means of a call to gp:registerimage-translation.

An image object

As returned by gp: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 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 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 large item images. The ${\tt image-function}$ 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-imagefunction 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. This restriction will be relaxed in a future release.

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.

See also image-list list-panel

make-image-locator

lower-interface Function

Summary The lower-interface function pushes the window contain-

ing a specified pane to the back of the screen.

Syntax 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 lower-interface.

See also interface

raise-interface
lower-interface
quit-interface

make-container

Generic Function

Summary The generic function make-container creates a container for

a specified element.

Syntax 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-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-general-image-set

Function

Syntax make-general-image-set &key width height count id

Description The make-general-image-set function creates an image set

object that refers to an image or a file containing an image.

The *id* keyword is a pathname or string identifying an image file, or a symbol previously registered with gp:registerimage-translation.

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

See also image-set

make-resource-image-set

make-image-locator

Function

Summary Creates an image locator object to use with toolbars, list

views and tree views.

Syntax make-image-locator &key image-set index

Description The function make-image-locator creates an image locator

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

image sets.

See also image-set

make-resource-image-set

Function

Syntax make-resource-image-set &key width height count library id

Description The make-resource-image-set function is only available on

the Windows implementation of LispWorks. It constructs an image set object that identifies an image stored as a bitmap

resource in a DLL.

The *library* keyword specifies a string giving the name of a DLL. The resource identifier of the bitmap is given by *id*.

The *width* and *height* are the dimensions of a single sub-image within the main image, and *count* specifies the number of

sub-images in the image.

See also image-set

make-general-image-set

map-collection-items

Generic Function

Summary The generic function map-collection-items calls a specified

function on all the items in a collection.

Syntax map-collection-items collection function

&optional collect-results-p

Description Calls function on each item in the collection by calling the col-

lection's items-map-function. If collect-results-p is non-nil,

the results of each call will be returned in a list.

Example (setq collection (make-instance 'capi:collection

:items '(1 2 3 4 5)))

(capi:map-collection-items collection

'princ-to-string t)

See also collection

choice

map-pane-children

Generic Function

Summary The map-pane-children generic function binds a variable to

each of the children of a layout.

Syntax map-pane-children (variable layout) &body body

Description This macro helps you to map over all of *layout*'s children, by

binding *variable* to each of the children across the body of code *body*. Note that this only works after *layout*'s description has been parsed, which generally happens just before a window is displayed on the screen, and the order in which the children are mapped is not necessarily the order that they are

placed into the pane's description.

```
Example
              (setg column (capi:contain
                              (make-instance
                               'capi:column-layout
                               :children (list
                                           (make-instance
                                            'capi:title-pane
                                            :text "Title 1")
                                           (make-instance
                                            'capi:title-pane
                                            :text "Title 2")
                                           (make-instance
                                            'capi:title-pane
                                            :text "A third title")
                                          ))))
              (capi:map-pane-children (pane column)
                                       (format t "~&~S: ~S"
                                        pane
                                        (capi:title-pane-text pane)))
See also
              layout
```

map-typeout Function

Syntax map-typeout pane &rest args

Description Makes a collector pane the visible child of a switchable lay-

out, 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

menu Class

Summary The class menu creates a menu for an interface when specified

as part of the menu bar (or as a submenu of a menu on the

menu bar).

Superclasses element

titled-menu-object

Slots items The items to appear in the menu.

items-function

A function to dynamically compute the

items.

Accessors menu-items

Description A menu has a title, and has items appearing in it, where an

item can be either a menu item, a menu component or

another menu.

The simplest way of providing items to a menu is to pass them using the item slot, but if you need to compute the items dynamically you should provide an items-function. This function gets called with the menu, and it should return a list of menu items for the new menu. This function is called before the popup-callback and the enabled-function which

means that they can affect the new items.

When debugging a menu, it may be useful to pop up a window containing a menu with the minimum of fuss. The func-

tion contain will do just that for you.

```
Examples
             (capi:contain (make-instance 'capi:menu
                                            :title "Test"
                                         :items '(:red :green :blue)))
             (capi:contain (make-instance 'capi:menu
                                            :title "Test"
                                          :items '(:red :green :blue)
                                            :print-function
                                             'string-capitalize))
              (capi:contain (make-instance
                             'capi:menu
                             :title "Test"
                             :items '(:red :green :blue)
                             :print-function 'string-capitalize
                             :callback #'(lambda (data interface)
                                            (capi:display-message
                                             "Pressed ~S" data))))
             Here is an example of how to add submenus to a menu
             (setq submenu (make-instance 'capi:menu
                                            :title "Submenu..."
                                            :items '(1 2 3)))
             (capi:contain (make-instance
                             'capi:menu
                             :title "Test"
                             :items (list submenu)))
             Finally, here is an example showing how to use the
             items-function:
             (capi:contain (make-instance
                             'capi:menu
                             :title "Test"
                             :items-function #'(lambda (menu)
                                                  (loop for i below 8
                                                   collect (random 10)
                                                  ))))
See also
             menu-component
             menu-item
             menu-object
             contain
```

menu-component

Class

Summary

The class menu-component is a choice that is used to group menu items and submenus both visually and functionally. The items contained by the menu-component appear separated from other items, menus, or menu components, by sep-

arators.

Superclasses choice

titled-menu-object

Slots The items to appear in the menu. items

items-function

A function to dynamically compute the items.

Description

Because menu-component is a choice, the component can be 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 in no selection components.

```
Example
```

```
(capi:contain (make-instance
              'capi:menu-component
              :items '(:red :green :blue)
              :print-function 'string-capitalize
              :interaction :single-selection))
(capi:contain (make-instance
               'capi:menu-component
               :items '(:red :green :blue)
               :print-function 'string-capitalize
               :interaction :multiple-selection))
```

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.

Superclasses

item

titled-menu-object

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.

Callbacks are made in response to a user gesture on a menu item. The callback type, function and data function (see callbacks) 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 call-back-type slot non-nil if needed.

Example

See also

menu-component

choice

menu

menu-object Class

Summary The class menu-object is the superclass of all menu objects,

and provides functionality for handling generic aspects of

menus, menu components and menu items.

Superclasses callbacks

Subclasses titled-menu-object

Slots enabled If nil, the menu object is disabled, and can-

not be selected.

popup-callback

Callback before the menu appears.

enabled-function

Returns non-nil if the menu is enabled.

enabled-slot The object is enabled if the slot is non-nil.

callback The selection callback for the object.

callback-data-function

A function to return data for the callback.

Accessors

menu-object-enabled menu-popup-callback

Description

When the menu object is about to appear on the screen, the CAPI does the following:

- 1. The popup-callback (if there is one) is called with the menu object and it is allowed to make arbitrary changes to that object.
- 2. The enabled function (if there is one) is called with the interface, and it should return non-nil if the menu object is enabled.
- **3.** The menu containing the object appears with all of the changes made.

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

The callback initary is placed in the selection-callback, extend-callback and retract-callback slots unless these are given explicitly, and so will get called when the menu object is selected or deselected. The

callback-data-function is a function that is called with no arguments and the value it returns is used as the data to the callbacks.

Example

See also menu

menu-item

menu-component

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.

Syntax modify-editor-pane-buffer pane

&key contents flag fill fixed-fill force

Description The modify-editor-pane-buffer function modifies the edi-

tor pane according to the keyword arguments. The argument *contents* (if non-nil) supplies a new string to place in the buffer. The *flag*, if given, sets the buffer-flag. If it is non-nil, *fill* causes the editor to fill each paragraph in the buffer. The argument *fixed-fill* sets the value of the slot fixed-fill and if non-nil causes the editor to fill each paragraph in the buffer.

See also editor-pane

mono-screen Class

Summary The mono-screen class is created for monochrome screen.

Superclasses screen

Description This is a subclass of screen that gets created for monochrome

screens. It is primarily available as a means of discriminating

on whether or not to use colors in an interface.

See also color-screen

multiple-line-text-input-pane

Class

Summary A pane allowing several lines of text to be entered.

Superclasses text-input-pane

Description The multiple-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 con-

tain newline characters.

See also text-input-pane

option-pane Class

Summary The class option-pane provides a pane which represents a

choice between a number of different items, and which only

displays the current one.

Superclasses choice

titled-pane

Slots enabled Non-nil if the option pane is enabled.

ACCESSORS option-pane-enabled

Description 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.

Example (setq option-pane (capi:contain

(make-instance 'capi:option-pane

:items '(1 2 3 4 5)

:selected-item 3)))

```
(setf (capi:choice-selected-item option-pane) 5)
(setf (capi:option-pane-enabled option-pane) nil)
(setf (capi:option-pane-enabled option-pane) t)
```

output-pane Class

Summary An output pane is a pane whose display and input behavior

can be controlled by the programmer.

Superclasses titled-pane

gp:graphics-port-mixin

Subclasses pinboard-layout

editor-pane

Slots display-callback

A function that knows how to redisplay the

pane.

input-model A list of input specifications.

scroll-callback

A function called when the user scrolls the

pane.

Accessors output-pane-display-callback

output-pane-scroll-callback

Readers output-pane-input-model

Description The class output-pane is a subclass of

gp:graphics-port-mixin which means that it supports many of the graphics ports drawing operations. When the CAPI needs to redisplay a region of the output pane, the display-callback gets called with the output-pane and the x, y, width and height of the region that needs redrawing, and the

callback should then use graphics port operations to redisplay that area. To force an area to be re-displayed, use the function gp:invalidate-rectangle.

The input-model provides a means to get callbacks on mouse and keyboard gestures. An input-model is a list of mappings from gesture to callback, where each mapping is a list of a gesture and a callback and optional arguments to be passed to the callback. The gesture itself is a list specifying the type of gesture, which can be a button, key, character or motion gesture, along with optional arguments specifying keyboard modifiers (:shift, :control and :meta) and actions associated with the gesture (:press, :release, :second-press and :motion). These options can be specified in any order, and if the list is only one item long then the gesture can be specified as just the item itself.

A button gesture should contain the button in question (either:button-1,:button-2 Or:button-3) along with an optional action (one of:press,:release,:second-press Or:motion) and zero or more keyboard modifiers.

A key gesture should contain the key in question (or the keyword :key meaning any key) along with an optional action (one of :press or :release) and zero or more keyboard modifiers.

A character gesture is a simple gesture, and can be either the character to be checked for, or the keyword :character meaning any character.

Finally, a motion gesture can either be defined in terms of dragging a button (in which case it is defined as a button gesture with the action :motion), or it can be defined for motions whilst no button is down by just specifying the keyword :motion with no additional arguments.

An input-model also accepts a command instead of a gesture, where a command is defined using define-command, and provides an alias for a gesture.

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 the user scrolls the output pane, the CAPI calls the scroll-callback with the output-pane and the new x and y coordinates of the top-left hand corner of the visible part of the output pane. This callback is for information only, as scrolling and redrawing any exposed areas will be handled by calls to the display-callback.

Examples

Firstly, here is an example that draws a circle in an output pane.

Here is an example that shows how to use a button gesture.

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)
             Finally, here is an example showing how to use the motion
             gesture.
             (capi:contain (make-instance
                             'capi:output-pane
                             :title "Drag button-1 across this pane:"
                             :input-model '(((:motion :button-1)
                                               gp:draw-point)))
                             :best-width 200 :best-height 200)
See also
             define-command
             pinboard-object
```

over-pinboard-object-p

Generic Function

Summary The over-pinboard-object-p generic function tests whether

a point lies within the boundary of a pinboard object.

Syntax over-pinboard-object-p $pinboard-object \times y$

Description 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 bound-

ing area of the pinboard object.

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.

Syntax page-setup-dialog &key screen owner printer

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.

The interface that owns the dialog is given by owner, and

defaults to the current top-level interface.

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.

Syntax pane-adjusted-offset pane adjust available-size actual-size

&key &allow-other-keys

Description This function calculates the offset required by the *adjust* key-

word 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 one of:

Place the pane at the top of the region.

:bottom Place the pane at the bottom of the region.

:left Place the pane at the left of the region.

:right Place the pane at the right of the region.

:center Place the pane in the center of the region.

However, new methods can accept alternative values for *adjust* where required and can also add extra keywords. For example, the <code>grid-layout</code> allows *adjust* to be a list of adjust values, and then passes the offset into this list as an additional keyword.

Example

```
(capi:pane-adjusted-offset button-panel :left 200 100)
```

100

See also layout

x-y-adjustable-layout

pane-adjusted-position

Generic Function

Summary

The pane-adjusted-position generic function calculates how to place a pane correctly within a layout, given a minimum and maximum position.

Syntax

pane-adjusted-position pane adjust min-position max-position &key &allow-other-keys

Description

This function calculates the position required by the *adjust* keyword 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.

Place the pane at the top of the region.

:bottom Place the pane at the bottom of the region.

:left Place the pane at the left of the region.

right Place the pane at the right of the region.

:center Place the pane in the center of the region.

However, new methods can accept alternative values for adjust where required and can also add extra keywords. For example, the <code>grid-layout</code> 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 <code>pane-adjusted-offset</code> as these changes will be seen by the default method of <code>pane-adjusted-position</code>.

Example

(setq button-panel (make-instance 'capi:button-panel

(capi:pane-adjusted-position button-panel :left 100 200)

See also layout

x-y-adjustable-layout

parse-layout-descriptor

Generic Function

Summary

The generic function parse-layout-description returns the geometry object associated with a layout's child.

Syntax parse-layout-descriptor child-descriptor interface layout

Description This generic function 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 chil-

dren in a layout.

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-

ble on the screen.

Superclasses text-input-pane

Description The password pane inherits all of its functionality from

text-input-pane, and thus it starts with the initial text and caret position specified by :text and :caret-position respectively, and limits the number of characters entered with the :max-characters keyword (which defaults to nil, meaning there is no maximum). It can be enabled and dis-

abled with the accessor text-input-pane-enabled.

Examples (setq password-pane (capi:contain

(make-instance
'capi:password-pane

:callback

"Password: ~A"
password)))))

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

See also editor-pane

text-input-pane

pinboard-layout

Class

Summary

The class pinboard-layout provides two very useful pieces of functionality for displaying CAPI windows. Firstly it is a layout that allows its children to be positioned anywhere within itself (like a pinboard). Secondly it supports pinboard-objects which are rectangular areas within the layout which have size and drawing functionality.

Superclasses output-pane

layout

Subclasses simple-pinboard-layout

Description When a p

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 :min-width and :max-width). The pinboard layout itself 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-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.

Example

Here is an example of a pinboard layout placing simple panes at arbitrary positions inside itself.

Here are some examples of the use of pinboard objects with pinboard layouts.

```
(capi:contain (make-instance
               'capi:pinboard-layout
               :description (list
                              (make-instance
                             'capi:image-pinboard-object
                             :image "new-lispworks-logo"
                               :x 20 :y 10)))
               :best-width 300 :best-height 300)
(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
                               :min-width 100)))
                :best-width 200 :best-height 200)
```

See also pinboard-object redraw-pinboard-object

pinboard-object Class

Summary

A pinboard object is a class that provides a rectangular area in a pinboard layout with drawing 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.

Superclasses element

Subclasses item-pinboard-object

image-pinboard-object
line-pinboard-object
drawn-pinboard-object

Slots output-pane The output pane on which the pinboard

object is drawn.

activep If t, the pinboard object is made active.

ACCESSORS pinboard-object-output-pane

pinboard-object-activep

Description 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).

There are a number of predefined pinboard objects provided

by the CAPI. They are as follows:

item-pinboard-object

Draws a title.

line-pinboard-object

Draws a line.

right-angle-line-pinboard-object

Draws a right-angled line.

image-pinboard-object

Draws an image.

titled-image-pinboard-object

Draws an image with a title.

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 <code>gp:invalidate-rectangle</code> on it (in which case the redrawing is done immediately), or call <code>redraw-pinboard-object</code> in which case the redrawing may be cached and displayed at a later date.

See also

pinboard-layout draw-pinboard-object graph-pane redraw-pinboard-object redraw-pinboard-layout

pinboard-object-at-position

Generic Function

Summary The generic function pinboard-object-at-position returns

the uppermost pinboard object containing a specified point.

Syntax pinboard-object-at-position $pinboard \ x \ y$

Description This function returns the uppermost pinboard object in the

pinboard that contains the point specified by *x* and *y*. It determines this by mapping over every pinboard object within the pinboard until it finds one for which the generic function

over-pinboard-object-p returns t.

Example (setq pinboard (capi:contain (make-instance

'capi:pinboard-layout)
:best-width 300
:best-height 300))

(make-instance 'capi:item-pinboard-object

:text "Hello world"
:x 100 :y 100
:parent pinboard)

(capi:pinboard-object-at-position pinboard 0 0)

(capi:pinboard-object-at-position pinboard 110 110)

See also over-pinboard-object-p

pinboard-object-overlap-p

pinboard-object
pinboard-layout

pinboard-object-overlap-p

Generic Function

Summary The generic function pinboard-object-overlap-p returns

non-mil if a specified region overlaps with the region of a

specified pinboard object.

Syntax pinboard-object-overlap-p pinboard-object

top-left-x top-left-y

bottom-right-x bottom-right-y

Description Returns non-nil if the specified region overlaps with the

region of the pinboard object.

See also pinboard-object-at-position

over-pinboard-object-p

pinboard-object
pinboard-layout

popup-confirmer

Function

Summary

The popup-confirmer function creates a dialog with predefined implementations of **OK** and **Cancel** buttons and a user specified pane in a layout with the buttons.

Syntax

popup-confirmer pane message

&rest interface-args

&key

value-function

ok-check ok-button no-button

cancel-button screen

exit-function ok-function no-function buttons callbacks callback-type

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.

The argument *value-function* should provide a callback which is passed *pane* and should return the value to return from <code>popup-confirmer</code>. If *value-function* is not supplied, then pane itself will be returned. If the *value-function* wants to indicate that the dialog cannot return a value currently, then it should return a second value that is non-nil.

The *ok-check* function is passed the result returned by the *value-function* and should return non-nil if it is acceptable for that value to be returned. These two functions are used by popup-confirmer to decide when the **OK** button should be enabled, thus stopping the dialog from returning with invalid data. The **OK** button's state can be updated by a call to redisplay-interface on the top-level, so the dialog should call it when the button may enable or disable.

The arguments ok-button, no-button and cancel-button are the text strings for the three buttons or nil meaning do not include such a button. The ok-button means return successfully from the dialog, 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.

The arguments exit-function, ok-function and no-function are the callbacks that get done when exiting, pressing **OK** and pressing **No** respectively. The exit-function defaults to exit-confirmer, the ok-function defaults to the exit-function and the no-function defaults to a function exiting the dialog with nil.

The arguments buttons, callbacks and callback-type are provided as a means of extending the available buttons. The buttons provided by buttons will be placed after the buttons generated by popup-confirmer, with the functions in callbacks being associated with them. Finally callback-type will be provided as the callback type for the buttons.

All other arguments will be passed to the call to make-instance for the interface that will be displayed using display-dialog. Thus geometry information, colors, and so on can be passed in here as well. By default, the dialog will pick up the foreground, background and font of pane.

Examples

Here are two simple examples which implement the basic functionality of two CAPI prompters: the first implements a simple prompt-for-string, while the second implements prompt-for-confirmation.

```
(capi:popup-confirmer
  (make-instance 'capi:text-input-pane
                 :callback
                 'capi:exit-confirmer)
  "Enter some text:"
  :value-function 'capi:text-input-pane-text)
(capi:popup-confirmer nil
  "Yes or no?"
  :callback-type :none
  :ok-button "Yes"
  :no-button "No"
  :cancel-button nil
  :value-function #'(lambda (dummy) t))
This example demonstrates the use of :redisplay-
interface to make the OK button enable and disable on each
keystroke.
(defun pane-integer (pane)
  (ignore-errors (values
                   (read-from-string
                    (capi:text-input-pane-text
                     pane)))))
(capi:popup-confirmer
  (make-instance 'capi:text-input-pane
                 :callback 'capi:exit-confirmer
                 :change-callback :redisplay-interface)
  "Enter an integer"
  :value-function 'pane-integer
  :ok-check 'integerp)
exit-confirmer
```

print-collection-item

See also

Generic Function

Summary The generic function print-collection-item prints an item

as a string using a specified print function.

Syntax print-collection-item item choice

display-dialog

Description This prints *item* as a string according to the print function in

choice.

Example (setq collection (make-instance

'capi:collection
:items '(1 2 3 4 5)

(capi:print-collection-item 2 collection)

See also get-collection-item

collection

print-dialog Function

Summary Displays a print dialog and returns a printer object.

Syntax print-dialog &key screen owner first-page last-page print-

selection-p print-pages-p print-copies-p

Description The print-dialog function displays a print dialog and

returns a printer object. The print 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* keyword 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.

The interface that owns the dialog is given by *owner*. It should be a currently displayed interface, and defaults to the current top-level interface.

See also

print-file
print-text

print-editor-buffer

Function

Summary Prints the contents of an editor buffer to the printer.

Syntax print-editor-buffer buffer &key start end printer interactive

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. If interactive is t, the default value, then a printer dia-

log is displayed.

See also print-file

print-text

print-file Function

Summary Prints the contents of a specified file.

Syntax print-file file &key printer interactive

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.

See also print-editor-buffer

print-text

print-text Function

Summary Prints plain text to a printer.

Syntax print-text line-function & key printer tab-spacing interactive

Description The print-text function prints plain text to a printer speci-

fied 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 mil when the text

is exhausted.

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

tered.

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

default behavior.

See also print-editor-buffer

print-file

printer-metrics-device-height

Function

Summary Returns the height of the printable page in the internal units

used by the printer driver or printing subsystem.

Syntax printer-metrics-device-height printer-metrics

Description

The printer-metrics-device-height function takes a printer-metric object as its argument and returns the height of the printable page in the internal units used by the printer driver or printing subsystem of the printer. This function and printer-metrics-device-width 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.

See also

get-printer-metrics

printer-metrics-device-width

printer-metrics-dpi-x printer-metrics-dpi-y printer-metrics-height printer-metrics-width

printer-metrics-device-width

Function

Summary

Returns the width of the printable page in the internal units

used by the printer driver or printing subsystem.

Syntax

printer-metrics-device-width printer-metrics

Description

The printer-metrics-device-width function takes a printer-metric object as its argument and returns the width of the printable page in the internal units used by the printer driver or printing subsystem of the printer. This function and printer-metrics-device-height 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.

See also

get-printer-metrics

printer-metrics-device-height

printer-metrics-dpi-x
printer-metrics-dpi-y
printer-metrics-height
printer-metrics-width

printer-metrics-dpi-x

Function

Summary Returns the number of printer device units per inch in the x

direction.

Syntax printer-metrics-dpi-x printer-metrics

Description The printer-metrics-dpi-x function returns the number of

printer device units per inch in the x direction. 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.

See also get-printer-metrics

printer-metrics-device-height
printer-metrics-device-width

printer-metrics-dpi-y
printer-metrics-height
printer-metrics-width

printer-metrics-dpi-y

Function

Summary Returns the number of printer device units per inch in the y

direction.

Syntax printer-metrics-dpi-y printer-metrics

Description The printer-metrics-dpi-y function returns the number of

printer device units per inch in the y direction. 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.

See also get-printer-metrics

printer-metrics-device-height
printer-metrics-device-width

printer-metrics-dpi-x
printer-metrics-height
printer-metrics-width

printer-metrics-height

Function

Summary Returns the height of the printable area in millimeters.

Syntax printer-metrics-height printer-metrics

Description The printer-metrics-height function takes a printer-met-

rics object and returns the height of the printable area in mil-

limeters.

See also get-printer-metrics

printer-metrics-device-height
printer-metrics-device-width

printer-metrics-dpi-x
printer-metrics-dpi-y
printer-metrics-width

printer-metrics-width

Function

Summary Returns the width of the printable area in millimeters.

Syntax printer-metrics-width printer-metrics

Description The printer-metrics-width function takes a printer-metrics

object and returns the width of the printable area in millime-

ters.

See also get-printer-metrics

printer-metrics-device-height
printer-metrics-device-width

printer-metrics-dpi-x
printer-metrics-dpi-y
printer-metrics-height

progress-bar Class

Summary A pane that is used to show progress during a lengthy task.

Superclasses range-pane

titled-pane

Description This pane is used to display progress during a lengthy task. It

has no interactive behavior.

The accessors (setf range-start) and (setf range-end) are used to specify the range of values the progress bar can

display.

The accessor (setf range-slug-start) is used to set the

progress indication.

See also range-pane

titled-pane

prompt-for-confirmation

Function

Summary The prompt-for-confirmation function pops up a dialog

box with a message and Yes and No buttons.

Syntax prompt-for-confirmation message &key screen

Description This pops up a dialog box containing message, with Yes and

No buttons. It returns two values:

- a boolean indicating whether Yes was pressed
- t (for compatibility with other prompt functions)

Example (capi:prompt-for-confirmation "Continue?")

See also confirm-yes-or-no

prompt-for-file Function

Summary The function prompt-for-file pops up a dialog prompting

the user for a file name.

Syntax prompt-for-file message akey pathname ok-check pane-args

popup-args filter filters if-exists if-does-not-exist operation owner

Description The function prompt-for-file prompts the user for a file

The function prompt-for-file prompts the user for a file using a dialog box, optionally defaulting to pathname. Like all the prompters, prompt-for-file returns two values: the file and a flag indicating success. The success flag will return mil if the dialog was cancelled, and to otherwise. An ok-check function can be specified, which should return non-mil if a given pathname is valid.

filter specifies the initial filter expression (defaults to "*.*"). filters is a plist of filter-names and filter expressions. If the filter argument is not one of the expressions in filters, an extra filter called "Files" is added for this expression. The default filters plist is:

```
`("Lisp Source Files" "*.LISP;*.LSP"
    "Lisp Fasls" "*.FSL"
    "Text Documents" "*.DOC;*.TXT"
    "Image Files" "*.BMP;*.DIB;*.ICO;*.CUR"
    "All Files" "*.*")
```

if-exists is one of :ok or :prompt. When set to :ok, the an existing file can be returned. Otherwise the user is prompted about whether the file can be overwritten. The default is :ok

when operation is :open and :prompt when operation is :save.

if-does-not-exist is one of :ok, :prompt or :error. When set to :ok, a nonexistent file can be chosen. When set to :prompt, the user is prompted if a non existent file is chosen. When set to :error, the user cannot choose a non existent file. The default is :prompt if operation is :open and :ok if operation is :save. operation is one of :open or :save. This chooses the style of dialog used. The default is :open. owner is a capi top-level interface which becomes the owner of the dialog. In a CAPI callback, this defaults to current interface. Otherwise an interface is chosen for the current thread.

Finally, as with all of the prompting functions, the prompt itself is created by passing an appropriate pane to popup-confirmer. Arguments can be passed to the make-instance of the pane and the call to popup-confirmer using pane-args and popup-args respectively. Currently, the pane used to create the file prompter is internal to the CAPI.

prompt-for-form

Function

Summary The prompt-for-form function pops up a text input pane and prompts the user for a form.

Syntax

prompt-for-form message

&key
package initial-value
evaluate quotify
ok-check value-function
pane-args popup-args

Description

This function 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 <code>package</code> if specified or <code>*package*</code> if not. If <code>evaluate</code> is non-nil then the result is the evaluation of the form, otherwise it is just the form itself. The printed version of <code>initial-value</code> will be placed into the text input pane as a default, unless <code>quotify</code>, which defaults to <code>evaluate</code>, specifies otherwise. If <code>value-function</code> is provided it overrides the default value function which reads the form and evaluates it when required. If the <code>ok-check</code> is provided it will be passed the entered form and should return <code>t</code> if the form is a valid result.

Finally, as with all of the prompting functions, the prompter is created by passing an appropriate pane (in this case a text input pane) to popup-confirmer. Arguments can be passed to the make-instance of the pane and the call to popup-confirmer using pane-args and popup-args respectively.

Example

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 The prompt-for-forms function pops up a text input pane

prompting the user for a number of forms.

Syntax prompt-for-forms message

&key

package initial-value value-function

pane-args popup-args

Description This function prompts the user for a number of forms by pro-

viding 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.

Finally, as with all of the prompting functions, the prompter is created by passing an appropriate pane (in this case a text input pane) to popup-confirmer. Arguments can be passed to

the make-instance of the pane and the call to

popup-confirmer using pane-args and popup-args respectively.

Example Try the following example, and enter 1 2 3 into the input

pane.

(capi:prompt-for-forms "Enter some forms:")

See also prompt-for-form

prompt-for-string
popup-confirmer
text-input-pane

prompt-for-integer

Function

Summary The prompt-for-integer function pops up a text input pane

and prompts the user for an integer.

Syntax prompt-for-integer message

&key

min max initial-value ok-check

pane-args popup-args

Description

This function prompts the user for an integer which is returned, and the *initial-value* will appear in the input pane. When min or max are specified the allowable result is constrained accordingly, and more complicated restrictions can be applied using the function ok-check which gets passed the currently entered number, and should return \mathbf{t} if it is valid.

Finally, as with all of the prompting functions, the prompter is created by passing an appropriate pane (in this case a text input pane) to popup-confirmer. Arguments can be passed to the make-instance of the pane and the call to popup-confirmer using pane-args and popup-args respectively.

Examples

```
(capi:prompt-for-integer "Enter an integer:")
(capi:prompt-for-integer "Enter an integer:" :max 10)
(capi:prompt-for-integer "Enter an integer:" :min 100 :max 200)
(capi:prompt-for-integer "Enter an integer:" :ok-check 'evenp)
```

See also

prompt-for-string
popup-confirmer
text-input-pane

prompt-for-string

Function

Summary The prompt-for-string function pops up a text input pane

and prompts the user for a string.

Syntax prompt-for-string message

&key

text initial-value print-function

value-function ok-check pane-args popup-args

Description This function prompts the user for a string and returns that

string and a flag indicating that the dialog was not cancelled. The initial string can either be passed in as text using the *text* argument, or by passing a value and a print-function for that value. If the print-function is not specified, then it will default to princ-to-string. The value returned can be converted into a different value by passing a *value-function*, which by default is identity. This *value-function* gets passed the text that was entered into the pane, and should return both the value to return and a flag that should be non-nil if the value that was entered is not acceptable. If an *ok-check* is passed, then it should return non-nil if the value about to be returned is acceptable.

Finally, as with all of the prompting functions, the prompter is created by passing an appropriate pane (in this case a text input pane) to popup-confirmer. Arguments can be passed to the make-instance of the pane and the call to popup-confirmer using pane-args and popup-args respectively.

Examples (capi:prompt-for-string "Enter a string:")

prompt-for-symbol

Function

Summary

The prompt-for-symbol function pops up a text input pane and prompts the user for a symbol. The symbols that are valid can be constrained in a number of ways.

Syntax

prompt-for-symbol message &key symbols package ok-check pane-args popup-args

Description

This function prompts the user for a symbol which they should type into a string prompter. The symbols that are valid can be constrained in a number of ways. The keyword arguments are a list of all valid symbols, a package in which the symbol must be available, and an *ok-check* function which when called on a symbol will return non-nil if the symbol is valid.

Finally, as with all of the prompting functions, the prompter is created by passing an appropriate pane (in this case a text input pane) to popup-confirmer. Arguments can be passed to the make-instance of the pane and the call to popup-confirmer using pane-args and popup-args respectively.

prompt-with-list

Function

Summary

The prompt-with-list function prompts the user to make a choice from a list. By default the choice is a list panel.

Syntax

prompt-with-list items message

&kev

choice-class interaction value-function pane-args popup-args

Description

This function prompts the user with a list of items from which they must select a certain number. By default, the user is prompted with a single selection list panel, but the class of pane (which must be a choice) can be specified using *choice-class*, and the interaction can be specified with *interaction*. Finally the value that is returned is usually the selected items, but a *value-function* can be supplied that gets passed the result and can then return a new result.

For single selection, the dialog has an **OK** and a **Cancel** button, while for other selections it has **Yes**, **No** and **Cancel** but-

tons where **Yes** means accept the selection, **No** means accept a null selection and **Cance**l behaves as normal.

Finally, as with all of the prompting functions, 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.

```
Examples
             (capi:prompt-with-list
                '(1 2 3 4 5) "Select an item:")
             (capi:prompt-with-list
                '(1 2 3 4 5) "Select some items:"
               :interaction :multiple-selection)
             (capi:prompt-with-list
               '(1 2 3 4 5) "Select an item:"
               :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)
             (capi:prompt-with-list
               '(1 2 3 4 5) "Select an item:"
               :interaction :multiple-selection
               :choice-class 'capi:button-panel
               :pane-args
                '(:layout-class capi:column-layout))
See also
             popup-confirmer
             list-panel
             choice
```

push-button Class

Summary

A push-button is a pane that displays either a piece of text or a generic image and when it is pressed it performs an action.

Superclasses button

titled-pane

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.

Example (setq button (capi:contain

(make-instance
'capi:push-button
:text "Press Me"
:data '(:some :data)

(setf (capi:button-enabled button) nil)
(setf (capi:button-enabled button) t)

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 but-

tons.

Superclasses button-panel

Description The class push-button-panel inherits all of its behavior from

button-panel, which itself inherits most of its behavior from

choice. Thus, the push button panel can accept items, callbacks, and so on.

```
Examples
             (defun test-callback (data interface)
                     (capi:display-message
                                       "Pressed ~S" data))
             (capi:contain (make-instance 'capi:push-button-panel
                                            :title "Press a button:"
                                             :items
                                               '("Press Me" "No, Me")
                                             :selection-callback
                                               'test-callback))
              (capi:contain (make-instance 'capi:push-button-panel
                                            :title "Press a button:"
                                            :items
                                               '("Press Me" "No, Me")
                                             :selection-callback
                                               'test-callback
                                            :layout-class
                                               'capi:column-layout))
             (capi:contain (make-instance 'capi:push-button-panel
                                             :title "Press a button:"
                                           :items '(1 2 3 4 5 6 7 8 9)
                                             :selection-callback
                                                'test-callback
                                             :layout-class
                                                'capi:grid-layout
                                             :layout-args
                                                '(:columns 3)))
See also
             push-button
             radio-button-panel
             check-button-panel
```

quit-interface Function

Summary The quit-interface function closes the top level interface

containing a specified pane.

Syntax quit-interface pane &key force

Description

This 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 okay to close the interface, it then calls destroy to do so. If force is specified, then neither the confirm-destroy-function or the destroy-callback are called, and the window is just closed immediately.

Example

Here are two examples demonstrating the use of quit-interface with the destroy-callback and the confirm-destroy-function.

(capi:quit-interface interface)

With this second example, the user is prompted as to whether or not to quit the interface.

CL-USER 4> (capi:quit-interface interface)

See also

destroy display interface radio-button Class

Summary A radio-button is a button that can be either selected or

deselected, but when selecting it any other buttons in its

group will be cleared.

Superclasses button

titled-pane

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 but-

ton's behavior.

Example (setq button (capi:contain

(setf (capi:button-selected button) t)

(setf (capi:button-selected button) nil)

(setf (capi:button-enabled button) nil)

(setf (capi:button-enabled button) t)

See also push-button

check-button button-panel

radio-button-panel

radio-button-panel

Class

Summary A radio-button-panel is a pane containing a group of but-

tons of which only one can be selected at any time.

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))

'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)

See also radio-button

push-button-panel
check-button-panel

range-pane Class

Summary This class exists to support the progress-bar and slider

classes. Consult the reference pages for these classes for fur-

ther information.

Superclasses None

Subclasses progress-bar

slider

Slots start The lowest value of the range.

end The highest value of the range.

slug-start The current value of the range.

callback Called when the user changes the value.

ACCESSORS range-start

range-end

range-slug-start range-callback

raise-interface Function

Summary The raise-interface function raises the interface containing

a specified pane to the front of the screen.

Syntax raise-interface pane

Description This 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 lower-interface.

Example (setq pane (capi:contain

(make-instance

'capi:text-input-pane)))

(capi:lower-interface pane)

(capi:raise-interface pane)

See also interface

raise-interface lower-interface quit-interface

redisplay-interface

Generic Function

Summary The redisplay-interface generic function updates the state

of an interface.

Syntax redisplay-interface interface

Description This method updates the state of an interface, such as

enabling and disabling menus, buttons, and so forth, that might have changed since the last call. When using this as a callback, you can use :redisplay-interface instead of the symbol, and then it will get passed the correct arguments

regardless of the callback type.

Note: This method is called by popup-confirmer to update its button's enabled state, and so it should be called when

state changes in a dialog.

See also interface

redisplay-menu-bar
redraw-pinboard-layout

display

redisplay-menu-bar

Function

Summary The function redisplay-menu-bar updates the menu bar of

an interface.

Syntax redisplay-menu-bar interface

Description This method updates the interface's menu bar, such that

menus become enabled and disabled as appropriate.

Compatibility

Note

This function has been superseded by redisplay-interface, which updates the menu bar, but also updates other state

objects such as buttons, list panels and so on.

See also interface

redisplay-interface

redraw-pinboard-layout

Function

Summary The redraw-pinboard-layout function redraws any pin-

board objects within a specified rectangle.

Syntax redraw-pinboard-layout pinboard

x y width height

&optional (redisplay t)

Description This causes any pinboard objects within the given rectangle

of the pinboard layout to get redrawn. If redisplay is set to nil, then the redisplay will be cached until a later date.

See also pinboard-object

redraw-pinboard-object

redraw-pinboard-object

Function

Summary The redraw-pinboard-object function redraws a specified

pinboard object.

Syntax redraw-pinboard-object object &optional (redisplay t)

Description This causes the pinboard object object to be redrawn, unless

redisplay is nil in which case the redisplay will be cached

until a later date.

See also pinboard-object

pinboard-layout

redraw-pinboard-layout

remove-capi-object-property

Function

Summary The remove-capi-object-property function removes prop-

erties from the property list of an object.

Syntax remove-capi-object-property object property

Description All CAPI objects contain a property list, similar to the symbol

plist. The functions capi-object-property and (setf capi-object-property) are the recommended ways of setting properties, and remove-capi-object-property is the

way to remove a property.

Example (setq pane (make-instance 'capi:list-panel :items '(1 2 3)))

(capi:capi-object-property pane 'test-property)

(setf (capi:capi-object-property pane 'test-property)

"Test")

(capi:capi-object-property pane 'test-property)

(capi:remove-capi-object-property pane 'test-property)

(capi:capi-object-property pane 'test-property)

See also capi-object-property

capi-object

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.

Superclasses line-pinboard-object

Slots 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 diago-

nally. The type of line 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 rightangled edges rather than diagonal ones.

```
Example
             (capi:contain (make-instance
                             'capi:right-angle-line-pinboard-object
                             :min-width 100
                             :min-height 100))
             (capi:contain (make-instance
                             'capi:right-angle-line-pinboard-object
                             :min-width 100
                             :min-height 100
                             :type :horizontal-first))
             (capi:contain (list
                              (make-instance
                               'capi:right-angle-line-pinboard-object
                               :min-width 100
                               :min-height 100)
                              (make-instance
                               'capi:right-angle-line-pinboard-object
                                :min-width 100
                                :min-height 100
                                :type :horizontal-first)))
```

row-layout Class

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

pinboard-layout

Superclasses grid-layout

See also

Slots ratios The size ratios between the layout's chil-

dren.

```
The vertical adjustment for each child.
              adjust
                              The gap between each child.
              gap
              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 layouts equivalent key-
              Words x-ratios, y-adjust, x-gap and x-uniform-size-p.
Examples
              (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)))))
              (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)))
              (setf (capi:layout-y-adjust row) :bottom)
              (setf (capi:layout-y-adjust row) :top)
```

See also column-layout

screen Class

Summary A screen is an object that represents each of the known mon-

itor screens.

Superclasses capi-object

Subclasses color-screen

mono-screen

Slots 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 screen object

per monitor screen, and they are then used to specify where a window is to appear. They 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.

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

(capi:screen-width screen)

(capi:screen-height screen)

(capi:display (make-instance

(capi:screen-interfaces screen)

See also convert-to-screen

search-for-item

Generic Function

Summary The generic function search-for-item returns the index of

an item in a collection.

Syntax 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. This is the counterpart function to get-collection-item which given an index, finds the

appropriate item.

See also get-collection-item

collection

set-button-panel-enabled-items

Generic Function

Summary The set-button-panel-enabled-items generic function sets

the enabled state of the items in a button panel.

Syntax set-button-panel-enabled-items button-panel

&key enable disable set test key

Description This function 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-geometric-hint

Function

Summary The set-geometric-hint function sets the hint associated

with a key.

Syntax 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 The set-hint-table function sets the hint table for an ele-

ment to be its property list.

Syntax set-hint-table element plist

Description Set the hint table for the *element* to be the *plist*. All existing

hints are retained for keys not in the plist.

See also set-geometric-hint

element

set-scroll-position

Generic Function

Summary The set-scroll-position generic function moves the scroll

bars of a given pane to a specified position.

Syntax set-scroll-position pane $x \ y$

Description Move the scroll bars of *pane* to place the given x and y coordi-

nates in the top left-hand corner of the visible window.

See also set-scroll-range

simple-pane

set-scroll-range

Generic Function

Summary The set-scroll-range generic function sets the range over

which the scroll bars for a pane apply.

Syntax set-scroll-range pane width height

Description Set the range over which the scroll bars for *pane* apply.

See also set-scroll-position

simple-pane

show-interface Function

Summary The show-interface function brings the interface containing

a specified pane back onto the screen.

Syntax 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

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).

Superclasses x-y-adjustable-layout

Description A simple layout's description can be either a single child, or a

list containing just one child. The simple layout then adopts the size constraints of its child, and lays the child out inside

itself.

Example (capi:contain (make-instance

'capi:simple-layout

:description (make-instance

'capi:text-input-pane)))

See also layout

row-layout column-layout

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.

Superclasses element

Subclasses titled-pane

Slots background The background color of the pane.

foreground The foreground color of the pane.

font The default font for the pane.

horizontal-scroll

Non-mil if the pane can scroll horizontally.

vertical-scroll

Non-mil if the pane can scroll vertically.

visible-border

Controls whether the pane has a border.

Accessors simple-pane-background

simple-pane-foreground

simple-pane-font

Readers simple-pane-horizontal-scroll

simple-pane-vertical-scroll simple-pane-visible-border

Description The background and foreground colors are colors specified

using the graphics ports color system, and the font must be a generic font. The value for visible-border can be any of the

following:

nil Has no border.

t Has a border.

:default Use the default for the window type.

Any simple pane can be made to scroll by specifying t to :horizontal-scroll or :vertical-scroll. By default these values are nil, but some subclasses of simple-pane default to t where appropriate (for instance editors always default to beging a position and these)

having a vertical scroll-bar).

In order to display a simple pane, it needs to be contained within an interface. The two convenience functions make-container and contain are provided to create an interface with enough support for that pane. The function make-container just returns a container for an element, and the function contain displays an interface created for the pane using make-container.

Examples (capi:contain (make-instance 'capi:simple-pane))

(capi:contain (make-instance 'capi:simple-pane

:background :red))

:horizontal-scroll t))

See also contain

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.

Superclasses pinboard-layout

simple-layout

Subclasses graph-pane

Slots 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
              (setg column (make-instance
                            'capi:column-layout
                            :description
                            (list
                             (make-instance
                              'capi:image-pinboard-object
                              :image "new-lispworks-logo")
                             (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.

Syntax simple-print-port port &key jobname scale dpi printer

interactive background

Description The simple-print-port function prints the output pane

specified by *port* to the default printer, unless specified otherwise by *printer*. The arguments of *scale* and *dpi* are used to determine how to transform the output pane's coordinate space to physical units. There meaning here is the same as in <code>get-page-area</code>, except that *scale* may also take the value <code>:scale-to-fit</code>, in which case the pane is printed as large as

possible on a single sheet.

The background color of the pane is ignored, and the value given by *background* is used instead. This defaults to :white.

If *interactive* is t, a print dialog is displayed. This is the default. If *interactive* is nil, then the document is printed to the current printer without prompting the user.

See also print-dialog

slider Class

Summary A pane with a sliding marker, which allows the user to con-

trol a numerical value within a specified range.

Superclasses range-pane

titled-pane

Description The slider class allows the user to enter a number by mov-

ing a marker on a sliding scale to the desired value.

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.

Superclasses simple-layout

Slots visible-child The currently visible child from the children.

ACCESSORS switchable-layout-visible-child

Description The switchable-layout is passed a list of children to be its

initial list of children, and also the initially visible child (which defaults to the first of the children). It inherits all of its layout behavior from simple-layout as it only ever lays out

one child at a time.

```
Example
             (setq children (list
                              (make-instance 'capi:push-button
                                              :text "Press Me")
                              (make-instance 'capi:list-panel
                                              :items '(1 2 3 4 5))))
             (setq layout (capi:contain
                            (make-instance
                             'capi:switchable-layout
                             :description children)))
             (setf (capi:switchable-layout-visible-child layout)
                    (second children))
              (setf (capi:switchable-layout-visible-child layout)
                    (first children))
See also
             layout
```

tab-layout Class

Summary The class tab-layout has two distinct modes. The first lays

out 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 the second mode the tabs are linked to a

The front. In the second mode the tabs are mixed to

:selection-callback as for button-panel.

Superclasses choice

layout

Slots description The main layout description.

items Specifies the tabs of the tab layout.

visible-child-function

The visible children for a given selection.

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.

selection-callback

The function called when a tab is selected.

Description

A tab layout can have two distinct modes. In its first 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 its second 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.

Examples

The following example shows the use of a switchable tab layout. Each tab is linked to an output pane by pairing them in the :items list.

Here is an example of the second mode of a tab layout, which uses the selection of a tab to change the nodes of a graph pane through the use of a 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
```

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.

Superclasses choice

text-input-pane

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 selection callback is called when the user selects an item using the drop-down list, or when the user presses the return

key.

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.

Superclasses titled-pane

Subclasses password-pane

Slots 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 Is the text-input-pane enabled?

completion-function

A function called when the user completes

the input by pressing tab.

callback-type The type of arguments to the callback.

callback A function called when the user presses

Return.

change-callback-type

The type of arguments to the callback.

change-callback

A function called when a change is made.

confirm-change-function

A function called to validate a change.

Accessors text-input-pane-text

text-input-pane-caret-position 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-enabled

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 :text and

:caret-position respectively, and limits the number of characters entered with the :max-characters keyword (which defaults to nil, meaning there is no maximum). The

pane can be enabled and disabled with the accessor

text-input-pane-enabled.

A completion-function can be specified which will get called when the completion gesture is made by the user (or pressing the Tab key). The function is called with the pane and the text to complete and should return either nil, the completed text or a list of possible completions. In the latter case, the CAPI will prompt the user for the selection they wish and this will become the new text.

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 callback can be specified with the change-callback-type.

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 okay to make the change. If nil is returned, then the pane will be unaltered (and a beep will be signalled indicating that the new values were invalid.

Compatibility Note

The confirm-change-function was called before-change-callback in LispWorks 3.1. Both the old initary and the old accessor are still supported, but may not be in future releases.

Examples

titled-image-pinboard-object

Class

Summary A titled-image-pinboard-object is a pinboard-object

that displays itself as a generic image with a title underneath

it.

Superclasses image-pinboard-object

item-pinboard-object

Slots title The title of the pinboard object.

x-adjust The offset adjustment between the image

and title.

Description The relative positions of the title and the image can be speci-

fied with adjust. By default, the adjust is :center, meaning that the image and title should be centered, but the value can take any of the arguments accepted by pane-adjusted-

offset.

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 compo-

nents and menu items.

Superclasses menu-object

Subclasses menu

menu-component
menu-item

Slots title The title for the object.

title-function

Returns the title for the object.

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 with the object when the menu is about to appear and which should return the title to use.

titled-pane Class

Summary The class titled-pane 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).

Superclasses simple-pane

Subclasses interface

layout title-pane display-pane text-input-pane

button
list-panel
button-panel
option-pane
contain

Slots title A title string for the current pane (or mil).

title-args Keyword arguments to the title make-

instance.

title-position

The position of the title.

title-adjust How to adjust the title relative to the pane.

message A message string for the current pane (or

nil).

message-args Keyword arguments to the message make-instance.

Accessors

titled-pane-title titled-pane-message

Description

The titled pane makes its decorations from title panes, and their look can be changed with the arguments title-args and message-args.

The message is always placed below the pane, but the title's position can be adjusted by specifying a 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.

Place the title above the pane.

:bottom Place the title below 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.

Examples

Try each of these examples to see some of the effects that titled panes can produce. Note that text-input-pane is a subclass of titled-pane, and that it has a default title-position of :left.

```
(capi:contain (make-instance 'capi:text-input-pane
                                            :title "Enter some text:"
                                            :title-position :top
                                            :title-adjust :center))
             (capi:contain (make-instance 'capi:text-input-pane
                                           :title "Enter some text:"
                                           :title-position :top
                                           :title-adjust :right))
              (capi:contain (make-instance 'capi:text-input-pane
                                            :message "A message"))
             (capi:contain (make-instance 'capi:text-input-pane
                                            :message "A message"
                                          :title "Enter some text:"))
              (capi:contain (make-instance 'capi:text-input-pane
                                           :title "Enter some text:"
                                           :title-args
                                            '(:foreground :red)))
See also
             title-pane
```

title-pane Class

Summary This class provides a pane that displays a piece of text.

Superclasses titled-pane

Slots 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 for a pane,

and so the class titled-pane is provided as a class that sup-

ports placing title panes around itself.

By default, title panes are constrained so that they cannot resize (that is, the values of max-width and max-height are t)

but this can be altered by using :max-width nil or

:max-height nil.

Examples (setq title-pane (capi:contain (make-instance 'capi:title-pane :text "This is a title pane")))

(setf (capi:title-pane-text title-pane) "New title")

toolbar Class

Summary This class provides a pane containing toolbar buttons and

panes.

Superclasses collection

simple-pane toolbar-object

Slots dividerp If t, a divider line is drawn above the tool-

bar, to separate it from the menu bar. The

default value is mil.

images A list of images, in one-to-one correspon-

dence 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 corre-

spondence with the items. Elements corresponding to toolbar-button items or toolbar-component items are ignored

button-width The width of the toolbar buttons. The

default is 24.

button-height The height of the toolbar buttons. The

default is 22.

image-width The width of buttons in the toolbar. The

default value is 16.

image-height The height of buttons in the toolbar. The

default value is 15.

default-image-set

An optional image setting object which can be used to specify images. See toolbarbutton and image-set for more details.

Description

The class toolbar inherits from collection, and therefore has a list of items. It behaves in a similar manner to push-button-panel, which inherits from choice.

The items keyword 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-panes, option-panes, and text-input-choices.

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.

All toolbar buttons within the item list behave as push buttons. However, toolbar button components may be single-selection or multiple-selection. See toolbar-component for further details.

See also

collection

push-button-panel
toolbar-component

toolbar-button Class

Summary This class is used to create instances of toolbar buttons.

Superclasses item

toolbar-object

Slots callback A function that is called when the toolbar

button is pressed.

image A slot specifying the image to use for the

toolbar button.

tooltip An optional string which is displayed when

the mouse moves over the button.

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, :images and :tooltips arguments to toolbar Or toolbar-component

can be used.

The image may be one of the following:

A pathname or string

This specifies the filename of a file suitable for loading with gp:load-image. Currently

this must be a bitmap file.

A symbol The symbol must have been previously reg-

istered by means of a call to gp:register-

image-translation.

An image object, as returned by gp:load-image.

An image locator object

This allows a single bitmap to be created which contains several button images side by side. See make-image-locator for more

information. On 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.

The image should be of the correct size for the toolbar. By default, this is 16 pixels wide and 15 pixels high.

See also make-image-locator

toolbar

toolbar-component

toolbar-component

Class

Summary A toolbar component is used to group several toolbar buttons

together. Each component is separated from the surrounding

components and buttons.

Toolbar components are choices, and may be used to implement toolbars on which groups of button have single-selec-

tion or multiple-selection functionality.

Superclasses toolbar-object

choice

Slots images A list of images, in one-to-one correspon-

dence with the items. Elements corresponding to toolbar-button items or toolbar-

component items are ignored

callbacks A list of callback functions, in one-to-one

correspondence with the items.Elements corresponding to toolbar-button items or

toolbar-component items are ignored

tooltips

A list of tooltip strings, in one-to-one correspondence with the items. Elements corresponding to toolbar-button items or toolbar-component items are ignored

default-image-set

An optional image-set object which can be used to specify images. See toolbar-button and image-set for more details.

Description

The class toolbar-component inherits from choice, and hence has a list of items. Its behavior is broadly similar to button-panel.

The :items keyword 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-panes, option-panes, and text-input-choices.

For items that are not toolbar buttons or toolbar components, a toolbar button is automatically created, using the appropriate elements of the <code>images</code>, <code>callbacks</code> and <code>tooltips</code> lists. If no image is specified, the item itself is used as the image. For more information on acceptable values for images, see <code>toolbar-button</code>.

See also

toolbar

toolbar-button

toolbar-object

Class

Summary

This is a common superclass of toolbar, toolbar-button and toolbar-component.

Superclasses None

Subclasses toolbar

toolbar-button toolbar-component

Slots enabled If t, the toolbar object is enabled.

ACCESSORS simple-pane-enabled

toolbar-object-enabled-function

Description Any toolbar object may be disabled, by setting its enabled

state 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 enable function specified. This is evaluated whenever update-toolbar is called. If it returns t, the toolbar object will be enabled; if it returns

nil, the object will be disabled.

See also toolbar

toolbar-button toolbar-component update-toolbar

top-level-interface

Generic Function

Summary Returns the top level interface containing a specified pane.

Syntax top-level-interface pane

Description Returns the top level interface that contains *pane*.

See also top-level-interface-p

interface element

top-level-interface-p

Generic Function

Syntax top-level-interface-p pane

Description Returns non-nil if *pane* is a top level interface.

See also top-level-interface

interface element

undefine-menu Macro

Syntax undefine-menu function-name &rest args

Description This function undefines a menu created with define-menu.

See also define-menu

menu

unhighlight-pinboard-object

Generic Function

Syntax unhighlight-pinboard-object pinboard object &key (redisplay

t)

Description This removes the highlighting from a pinboard object if nec-

essary, 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

unmap-typeout Function

Syntax 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-pinboard-object

Function

Syntax update-pinboard-object object

Description This function checks the *object*'s constraints, and adjusts the

 $\it object$'s size as necessary. It then forces the layout to redisplay the $\it object$ at its new size. Finally, it returns t if a resize was

necessary.

See also redraw-pinboard-object

pinboard-object

update-toolbar Function

Summary Updates a toolbar object.

Syntax 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 mil.

See also toolbar

toolbar-button toolbar-component

with-atomic-redisplay

Macro

Summary The with-atomic-redisplay macro delays the updating of a

pane until all state changes have been performed.

Syntax with-atomic-redisplay (pane) &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 the pane until the end of the scope of the macro.

Compatibility

Note

The *pane* argument was new in LispWorks 3.2, and hence code written for 3.1 will have to be changed to supply it.

See also display

simple-pane

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.

Syntax with-document-pages page-var first-page last-page &body body

Description The with-document-pages evaluates body repeatedly, with

page-var bound to the number of the page to print on each iteration. It is used to by applications providing Page on

Demand printing.

The *first-page* and *last-page* arguments are evaluated to yield the page numbers of the first and last pages in the document.

See also with-page

with-print-job

with-geometry Macro

Summary The with-geometry macro is used for defining layouts and

for creating new pinboard-object classes, by binding a set of

variables to a pane's geometry.

Syntax with-geometry pane &body body

Description The macro 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. Its main uses are for defining layouts and for creating new

pinboard-object classes.

x The x position of the pane.

%y% The y position of the pane.

%width% The width in pixels of the pane.

%height% The height in pixels of the pane.

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

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

 ${\rm max-width}$ The maximum width of the pane.

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

%object% The object whose geometry this is.

%child% The same as %object% (kept for 3.1 compati-

bility).

%ratio% Ratio information.

Compatibility Note In LispWorks 3.1, this macro and all of its variable names were in the capi-layouts package. They have been moved

into the CAPI package for simplicity.

See also

element

with-page Macro

Summary Binds a variable to either t or mil, and executes a body of

code to print a page only if the variable is t.

Syntax with-page var &body body

Description The with-page macro binds var to t if a page is to be printed,

or nil if it is to be skipped. The *body* is executed once, and is expected to draw the document only if *var* is t. Each call to

with-page contributes a new page to the document.

See also with-document-pages

with-page-transform

with-page-transform

Macro

Summary Defines a rectangular region within the coordinate space of

an output pane or printer port.

Syntax 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.

Syntax with-print-job var &key pane jobname printer &body body

Description The with-print-job macro creates a print job which prints to

printer. If *printer* is not specified, the default printer is used. The macro binds *var* to a graphics port object, and printing is performed by using graphics port objects to draw the object.

If pane is specified it must be an output pane. In this case var is bound to pane, and the output pane is modified within the dynamic extent of the with-print-job so all drawing operations draw to the printer instead of the output pane. This can be useful when implementing printing by modifying existing redisplay code that is written expecting the output pane.

See also with-document-pages

with-page

with-page-transform

with-random-typeout

Macro

Summary Binds a stream variable to a collector pane.

Syntax with-random-typeout (stream-variable pane) &body body

Description The with-random-typeout macro 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

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).

Superclasses layout

Subclasses simple-layout

grid-layout

Slots 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 for x-adjust and y-adjust are used by layouts to

decide what to do when a pane is smaller than the space in which it is being laid out. The values themselves are interpreted by the function pane-adjusted-offset, which by

default can be any of the following:

Place the pane at the top of the region.

Place the pane at the bottom of the region. :bottom Place the pane at the left of the region. :left Place the pane at the right of the region. :right Place the pane in the center of the region. :center Example Note: column-layout is a subclass of x-y-adjustablelayout. (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)))))) (setf (capi:layout-x-adjust column) :right) (setf (capi:layout-x-adjust column) :center) See also pane-adjusted-offset

GP Reference Entries

The following chapter provides reference entries for the functions and macros exported from the gp package. You can use these functions to draw graphics in CAPI output panes, which are a kind of graphics port. See the Graphics Ports chapter in the *CAPI User Guide* for more information on graphics ports and their associated types.

default-image-translation-table

Variable

Summary The default image translation table.

Signature *default-image-translation-table*

Description The *default-image-translation-table* variable contains

the default image translation table. It is used if no image translation table is specified in calls to image translation table

functions.

unit-transform Variable

Summary The list (1 0 0 1 0 0).

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.

analyze-external-image

Function

Summary Returns the width, height, color-table, and number of impor-

tant colors for an external image.

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 an

external image.

apply-rotation Function

Summary Modifies a transform such that a rotating of a given number

of radians is performed on any points multiplied by the

transform.

Signature apply-rotation transform theta =>

Arguments transform A transform

theta A real number

Description The apply-rotation function modifies transform such that a

rotation of *theta* radians is performed on any points multiplied by the transform. Any operations already contained in

the transform occur before the new rotation.

apply-scale Function

Summary Modifies a transform such that a scaling occurs on any points

multiplied by the transform.

Signature apply-scale transform sx sy =>

Arguments transform A transform

sx A real number

sy An integer

Description The apply-scale function modifies transform such that a scal-

ing of *sx* in *x* and *sy* in *y* is performed on any points multiplied by the transform. Any operations already contained in

the transform occur before the new scaling.

apply-translation

Function

Summary Modifies a transform such that a translation is performed on

any points multiplied by the transform.

Signature apply-translation transform dx dy =>

Arguments transform A transform

dx A real number

dy A real number

Description The apply-translation function modifies transform such

that a translation of (dx dy) is performed on any points multiplied by the transform. Any operations already contained in

the transform occur before the new translation.

augment-font-description

Function

Summary Returns a font description combining the attributes of a given

font description with a set of font attributes.

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*. 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.

clear-external-image-conversions

Function

Summary Clears external image conversions for a port.

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

Description The clear-external-image-conversions function clears

the external image conversions for a port. If *gp-or-null* is nil all conversions are cleared using the image-color-users. If *all* is non-nil all conversions for all ports are cleared using *gp-or-null*. Conversions are also freed if *free-image* is non-nil. By default, *free-image* is t, *all* is (null *gp-or-null*), and *errorp* is t.

clear-graphics-port

Function

Summary Draws a filled rectangle covering the entire port in the port's

background color.

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.

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.

Signature clear-rectangle port x y width height =>

Arguments port A graphics port

x A real number

y A real number

width A real number

height A real number

Description The clear-rectangle function draws the rectangle specified

by x, y, width, and height in port's background color. All other

graphics state parameters are ignored.

compress-external-image

Function

Summary Converts external image data into compressed DIB format.

Signature compress-external-image external-image =>

Arguments external-image An external image

Description The compress-external-image function converts the exter-

nal-image data into compressed DIB format.

compute-char-extents

Function

Summary Returns the starting *x* coordinates of each of the characters in

a string if the string was printed to a graphics port.

Signature compute-char-extents port string &optional font => extents

Arguments *port* A graphics port

string A string
font A font

Values extents An array of integers

Description Returns the *extents* of the characters in *string* in the font asso-

ciated with port, or the font given. The extents are an array, one element per character, which gives the starting x coordi-

nate of that character if the string was drawn to port.

convert-external-image

Function

Summary Returns an image derived from an external image format.

Signature convert-external-image gp external-image &key cache force-new => image

Arguments gp A graphics port

external-image An external image

cache A boolean

force-new A boolean

Values image An image

Description The convert-external-image function returns an image

derived from *external-image*. The image is ready for drawing to the given graphics port. If *cache* is non-nil images conversions are cached by (color-device *a b*) in the *external-image* where *a* is the transparent-pixel-index and *b* the image-background-color of the port. If *force-new* is non-nil a new image is always created, and put in the cache. By default, *cache* has

the value *cache-external-images-p*.

copy-external-image

Function

Summary Returns a copy of an external image.

Signature copy-external-image external-image &key new-color-table =>

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.

Signature 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

Description The copy-pixels function copies a rectangular area from one

port to another. The transform, mask, mask-x and mask-y from the *to-port*'s graphics state are used. The (*to-x to-y*) is transformed according to *to-port*'s transform, but the image is not scaled or rotated. 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.

copy-transform

Function (inline)

Summary Returns a copy of a transform.

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.

Signature create-pixmap-port pane width height &key background collect relative clear => pixmap-port

Arguments pane A graphics port for a window

width An integer height An integer

background A color designator

collect A boolean
relative A boolean
clear A list or t

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 \mathfrak{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 \mathtt{nil} .

If *relative* is non-nil, the pixmap graphics port collects pixel coordinates corresponding to the left, top, right, and bottom extremes of the drawing operations taking place within the body forms, and if these extend beyond the edges of the pixmap (into negative coordinates for example) the entire drawing is offset by an amount which ensures it remains within the port. It is as if the port moves its relative origin in order to

accommodate the drawing. If the drawing size is greater than the screen size, then some of it is lost. The default value is nil.

If *collect* is non-nil, this causes the drawing extremes to be collected but without having the pixmap shift to accommodate the drawing, as *relative* does. The extreme values can be read using the get-bounds function, and make-image-fromport.

define-font-alias Function

Summary Defines an alias for a font.

Signature define-font-alias keyword font =>

Arguments keyword A keyword

font A font

Description The function define-font-alias defines keyword as an alias

for font.

destroy-pixmap-port

Function

Summary Destroys a pixmap port, thereby freeing any window system

resources it used.

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.

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 cur-

rent set of dithers is used in the decision.

See also initialize-dithers

with-dither

draw-arc Function

Summary Draws an arc.

Signature draw-arc port x y width height start-angle sweep-angle &rest

args &key filled =>

Arguments *port* A graphics port

x A real number

y A real number

width A real number

height A real number

start-angle A real number

sweep-angle A real number

filled A boolean

args General graphics port drawing arguments

Description

The draw-arc function draws an arc contained in the rectangle from (x y) to (x+width y+width) from start-angle to start-angle+sweep-angle. Both angles are specified in radians. Currently, arcs are parts of ellipses whose major and minor axes are parallel to the screen axes. If the port has rotation in its transform, the enclosing rectangle is modified to be the external enclosing orthogonal rectangle of the rotated rectangle. The start angle is rotated. The transform, foreground, background, operation, thickness, scale-thickness, stipple, pattern, mask-x, mask-y, and mask from the port's graphics state are all used. When filled is non-nil, a sector is drawn.

draw-arcs Function

Summary Draws several arcs.

Signature draw-arcs port description &rest args &key filled =>

Arguments *port* A graphics port

description A description sequence

filled A boolean

args General graphics port drawing arguments

Description The draw-arcs function draws several arcs as specified by

the *description* sequence. This is usually more efficient than making several calls to <code>draw-arc</code>. The *description* argument is a sequence of values of the form *x y width height start-angle*

sweep-angle. See draw-arc for more information.

See also draw-arc

draw-character Function

Summary Draws a character in a given graphics port.

Signature draw-character port character x y &rest args &key block =>

Arguments *port* A graphics port

character A character

x A real numbery A real number

block A boolean

args General graphics port drawing arguments

Description The draw-character function draws the character at $(x \ y)$ on

the port. The transform, foreground, background, operation, stipple, pattern, mask, mask-x, mask-y and font from the *port*'s graphics state are all used. The (x y) specifies the leftmost point of the character's baseline. *block*, if non-nil, causes the character to be drawn in a character cell filled with

the port's graphics state background.

draw-circle Function

Summary Draws a circle.

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 General graphics port drawing 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, thickness, scale-thickness, and mask from the port's graphics state are all used. When *filled* is non-ni1, the

circle is filled with the foreground color.

Examples (gp:draw-circle port 100 100 20)

(gp:draw-circle port 100 100 50 :filled t

:foreground :green)

draw-ellipse Function

Summary Draws an ellipse.

Signature draw-ellipse port x y x-radius y-radius &rest args &key filled

=>

Arguments *port* A graphics port.

x A real number.

y A real number.

x-radius A real number.y-radius A real number.

radius A real number.

args General graphics port drawing arguments.

filled A boolean.

Description The draw-ellipse function draws an ellipse of the given

radii centered on $(x\ y)$. The transform, foreground, background, operation, thickness, scale-thickness, and mask from the port's graphics state are all used. When *filled* is non-ni1,

the ellipse is filled with the foreground color.

Examples (gp:draw-ellipse port 100 100 20 40)

(gp:draw-ellipse port 100 100 50 10 :filled t

:foreground :green)

draw-image Function

Summary Displays an image on a graphics port at a given position.

Signature draw-image image to-x to-y &rest args &key from-x from-y to-width to-height from-width from-height =>

Arguments image An image

from-width

to-x A real number

to-y A real number

from-x A real number

from-y A real number

to-width A real number

to-height A real number

from-height A real number

args General graphics port drawing arguments

Description The draw-image function displays *image* on the port at *to-x to-*

A real number

y. Graphics state translation is guaranteed to be supported. Support for scaling and rotation are library dependent. The

default value of *from-x* and *from-y* is 0. The *width* and *height* arguments default to the size of the image.

draw-line Function

Summary Draws a line between two given points.

Signature draw-line port from-x from-y to-x to-y &rest args =>

Arguments *port* A graphics port

from-x A real number
from-y A real number
to-x A real number
to-y A real number

args General graphics port drawing arguments

Description The draw-line function draws a line from (from-x from-y) to

(to-x to-y). The graphics state parameters transform, fore-ground, background, operation, thickness, scale-thickness, dashed, dash, line-end-style, stipple, pattern, mask-x, mask, and mask are

used.

draw-lines Function

Summary Draws several lines between pairs of two given points.

Signature draw-lines port description &rest args =>

Arguments *port* A graphics port

description A description sequence

args General graphics port drawing arguments

Description The draw-line function draws several lines as specified by

the *description* sequence. This is usually more efficient than making several calls to draw-line. The *description* argument is a sequence of values of the form x1 y1 x2 y2. See draw-line

for more information.

See also draw-line

draw-point Function

Summary Draws a pixel at a given point.

Signature draw-point port x y &rest args =>

Arguments *port* A graphics port

x A real number

y real number

args General graphics port drawing arguments

Description The draw-point function draws a single-pixel point at (x y).

The transform, foreground, background, operation, stipple, pattern, mask-x, mask-y, and mask slots of the graphics state are

used.

draw-points Function

Summary Draws a pixel at a given point.

Signature draw-point port description &rest args =>

Arguments *port* A graphics port

description A description sequence

args General graphics port drawing arguments

Description The draw-point function draws several single-pixel points as

specified by the *description* argument, which is a sequence of *x y* pairs. It is usually faster than several calls to draw-point.

See draw-point for more information.

See also draw-point

draw-polygon Function

Summary Draws a polygon.

Signature draw-polygon port points &rest args &key filled closed fill-rule

=>

Arguments *port* A graphics port

points A description sequence

filled A boolean

closed A boolean

fill-rule A keyword

args General graphics port drawing arguments

Description The draw-polygon function draws a polygon using alternat-

ing 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, stipple, pattern, mask-x, mask-y, and mask from the port's graphics state are all used. The fill-rule specifies how overlapping regions are filled. Possible values are :even-odd and

:winding.

draw-polygons Function

Summary Draws several polygons.

Signature draw-polygon port description &rest args &key filled closed fill-

rule =>

Arguments *port* A graphics port

description A list of real numbers

filled A boolean

closed A boolean

fill-rule A keyword

args General graphics port drawing arguments

Description

The draw-polygons function draws several polygon using a sequence alternating x and y values in the *description* argument as 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*, *stipple*, *pattern*, *mask-x*, *mask-y*, and *mask* from the *port*'s graphics state are all used. The *fill-rule* specifies how overlapping regions are filled. Possible values are :even-odd and :winding.

Example

This draws two heavgons, one inside the other:

See also draw-polygon

draw-rectangle Function

Summary Draws a rectangle.

Signature draw-rectangle port x y width height &rest args &key filled =>

Arguments *port* A graphics port

x A real number

y A real number

width A real number

height A real number

filled A boolean

args General graphics port drawing arguments

Description

The draw-rectangle function draws a rectangle whose corners are (x y), (x+width y), (x+width y+height) and (x y+height). The filled keyword if non-nil causes a filled rectangle to be drawn. While the exact results are host-specific, it is intended that a filled rectangle does not include the lines (x = x+width) and (y = y+height) while a non-filled rectangle does. This function works correctly if the port's transform includes rotation. The graphics state parameters transform, foreground, background, operation, thickness, scale-thickness, dashed, dash, line-joint-style, stipple, pattern, mask-x, mask-y, and mask are used.

draw-rectangles

Function

Summary Draws several rectangles.

Signature draw-rectangle port description &rest args &key filled =>

Arguments *port* A graphics port

description A description sequence

filled A boolean

args General graphics port drawing arguments

Description

The draw-rectangles function draws several rectangles as specified in *description* which consists of a group of values given as x y width height. The filled keyword if non-nil causes filled rectangles to be drawn. While the exact results are host-specific, it is intended that a filled rectangle does not include the lines (x = x + width) and (y = y + height) while a non-filled rectangle does. This function works correctly if the port's transform includes rotation. The graphics state parameters transform, foreground, background, operation, thickness, scale-thickness, dashed, dash, line-joint-style, stipple, pattern, mask-x, mask-y, and mask are used.

See also draw-rectangle

draw-string Function

Summary Draws a string with the baseline positioned at a given point.

Signature draw-string port string x y &rest args &key start end block =>

Arguments port A graphics port

string A string

x A real numbery A real numberstart A real number

end A real number

block A boolean

args General graphics port drawing arguments

Description

Draws the string with the baseline starting at (x y). The transform, foreground, background, operation, stipple, pattern, mask, mask-x, mask-y and font from the port's graphics state are all used. *start* and *end* specify which elements of the *string* to draw. *block* if non-nil causes each character to be drawn in a character cell filled with the port's graphics state background.

By default, start is 0.

external-image Class

Summary A class representing a color image.

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.

external-image-color-table

Function

Summary Returns a vector containing RGB color specifications of an

external image.

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.

Signature (setf (external-image-color-table external-image)

replacement-color-table) =>

Arguments external-image An external image

replacement-color-table

A color table

Values

Description The external-image-color-table setf expander replaces the

color table in *external-image*. The color table specified by *replacement-color-table* must be the same length as the external image's original color table. It is a vector of RGB color-speci-

fications.

externalize-image

Function

Summary Returns an external image containing color information from

an image.

Signature externalize-image gp image &key maximum-colors

important-colors &allow-other-keys =>

external-image

Arguments gp A graphics port

image An image

maximum-colors An integer or nil important-colors An integer or nil

Values external-image An external image

Description

The external-image function returns an external image containing color information from *image*. If *maximum-colors* is nil, the default value, 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 the Color System parameters color::*local-rgbcolor-distance-red-weight* and so on 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.

find-best-font Function

Summary Returns the best font for a graphics port.

Signature find-best-font pane fdesc => font

Arguments pane A graphic port

fdesc A font description

Values font A font

Description The find-best-font function returns the best font for pane

which matches *fdesc*. When there alternative fonts available the choice of best font is operating system dependent.

find-matching-fonts

Function

Summary Returns a list of the font objects available for a pane.

Signature find-matching-fonts pane fdesc => fonts

Arguments pane A graphics port

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.

font-description

Function

Summary Returns a font description object for a given font.

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.

font-description-attributes

Function

Summary Returns the attributes of a given font description.

Signature font-description-attributes fdesc => font-attributes

Arguments *fdesc* A font description

Values font-attributes A list of font attributes

Description The font-description-attributes function returns the

attributes of the fdesc. The list should not be destructively

modified.

font-description-attribute-value

Function

Summary Returns the values of a given font attribute in a font descrip-

tion.

Signature font-description-attribute-value fdesc font-attribute =>

value

Arguments fdesc A font description

font-attribute A font attribute

Values value A font attribute value

Description The font-description-attribute-value function returns

the value of font-attribute in fdesc, or :wild if font-attribute is

not specified in fdesc.

font-fixed-width-p

Function

Summary Returns t if a specified font is of a fixed width.

Signature font-fixed-width-p port &optional font => bool

Arguments *port* A graphics port

font A font

Values bool A boolean

Description The font-fixed-width-p function returns t if the font associ-

ated with *port*, or the optionally specified *font*, is fixed width.

free-image Function

Summary Frees the library resources allocated with an image.

Signature free-image port image =>

Arguments *port* A graphics port

image An image

Description The free-image function frees the library resources associ-

ated with image. This should be done when an image is no

longer needed.

get-bounds Function

Summary Returns the four values of the currently collected drawing

extremes.

Signature get-bounds pixmap-port => left top right bottom

Arguments *pixmap-port* A graphics port

Values *left* An integer

top An integer right An integer

bottom An integer

Description The get-bounds functions returns the four values *left*, *top*,

right, bottom of the currently collected drawing extremes. The

values can be used to get an image from the port.

EXample (with-pixmap-graphics-port (p1 pane width height

:relative t)
(with-graphics-rotation (p1 0.123)

(draw-rectangle pl 100 100 200 120 :filled t

:foreground :red)

(get-bounds p1)))

produces the following output:

72 112

285

255

See also make-image-from-port

get-character-extent

Function

Summary Returns the extent of a character in pixels.

Signature get-character-extent port character & optional font =>

left, top, right, bottom

Arguments port A graphics port

character A character

font A font

Values *left* An integer

top An integer right An integer

bottom An integer

Description The get-character-extent function returns the extent in

pixels of the character in the font associated with port, or the

font given.

get-char-ascent

Function

Summary Returns the ascent of a character in pixels.

Signature get-char-ascent port character font => ascent

Arguments port A graphics port

character A character

font A font

Values ascent An integer

Description The get-character-ascent function returns the ascent in

pixels of the *character* in the font associated with *port*, or the

font given.

get-char-descent

Function

Summary Returns the descent of a character in pixels.

Signature get-char-descent port character font => descent

Arguments port A graphics port

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.

Signature get-char-width port character font => width

Arguments *port* A graphics port

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.

Signature x = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y = x + y

Arguments *points* Real numbers

Values x A real number

y A real numberwidth A real number

height 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.

Signature get-font-ascent port &optional font => ascent

Arguments *port* A graphics port

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.

Signature get-font-average-width port &optional font => average-width

Arguments *port* A graphics port

font A font

Values average-width An integer

Description The get-font-average-width function returns the average-

width in pixels of the font associated with port, or the font

given.

get-font-descent

Function

Summary Returns the descent in pixels of a font.

Signature get-font-descent port &optional font => descent

Arguments port A graphics port

font A font

Values *descent* An integer

Description The get-font-descent function returns the *descent* in pixels

of the font associated with *port*, or the *font* given.

get-font-height

Function

Summary Returns the height of a font.

Signature get-font-height port & optional font => height

Arguments *port* A graphics port

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.

Signature get-font-width port &optional font => width

GP Reference Entries

Arguments *port* A graphics port

font A font

Values width An integer

Description The get-font-width function returns the width in pixels of

the font associated with *port*, or the *font* given.

get-graphics-state

Function

Summary Returns the graphics state object for a graphics port.

Signature get-graphics-state port => state

Arguments port A graphics port

Values state A graphics state object

Description The get-graphics-state function returns the graphics state

object for port. The individual slots can be accessed using the

accessor functions.

See also make-graphics-state

get-origin Function

Summary Returns the coordinate origin of a pixmap graphics port.

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

get-string-extent

Function

Summary Returns the extent in pixels of a string.

Signature get-string-extent port string &optional font => left, top, right, bottom

Arguments port A graphics port

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.

get-transform-scale

Function

Summary Returns the overall scaling factor of a transform.

Signature get-transform-scale transform => result

Arguments transform A transform

Values result A real number

Description The get-transform-scale function returns a single number

representing the overall scaling factor present in the trans-

form.

graphics-port-transform

Function

Summary Returns the transform object of a graphics port.

Signature graphics-port-transform port => transform

Arguments *port* A graphics port

Values transform A transform object

Description The graphics-port-transform function returns the trans-

form object (a six-element list) associated with *port*.

image Class

Summary An abstract image object. An image can be drawn via draw-

image.

Superclasses

Subclasses

Description The image class is the abstract image object class. An image

can be drawn using draw-image.

See also draw-image

image-freed-p Function

Summary Determines whether an image has been freed.

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 mil otherwise.

image-loader Function

Summary Returns the image load function

Signature image-loader image-id &key image-translation-table => loader

Arguments image-id An image identifier

image-translation-table

An image translation table

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*.

image-translation

Function

Summary Returns the translation for an image registered in its image

translation table.

Signature image-translation image-id &key image-translation-table =>

translation

Arguments image-id An image identifier

image-translation-table

An image translation table

Values translation A translation

Description The image-translation function returns the translation for

image-id registered in image-translation-table. The default
value of image-translation-table is *default-image-

translation-table*.

initialize-dithers Function

Summary Initialize dither objects up to a given order.

Signature initialize-dithers & optional order =>

Arguments *order* An integer

Values

Description The initialize-dithers function initializes dither objects

up to the given *order* (size = $2 ^ order$). By default, order is 3.

inset-rectangle

Function (inline)

Summary Moves the corners of a rectangle inwards by a given amount.

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

Values

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.

Signature inside-rectangle rectangle $x \ y \Rightarrow result$

Arguments rectangle A list of integers

x An integery An integer

Values result A boolean

Description

The inside-rectangle function returns t if the point (x y) is inside rectangle. The rectangle is expected to be ordered; if the rectangle is specified by (left right top bottom), then left must be less than right, and bottom must be less than top. The lines y = bottom and x = right are not considered to be inside the rectangle.

invert-transform Function

Summary Constructs the inverse of a transform.

Signature invert-transform transform transform

Arguments transform A transform

into A transform or nil

Values inverse A transform

Description This function constructs the inverse of *transform*. If *T* is *trans-*

form and T is its inverse, then TT = I. If into is non-nil it is modified to contain T and returned, otherwise a new trans-

form is constructed and returned.

load-image Function

Summary Loads an image and returns the image object.

Signature load-image gp id &key cache image-translation-table => image

Arguments gp A graphics port

id An image identifier

cache A keyword

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 <code>image</code> 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.

The *cache* argument controls whether the image translation is cached. See the convert-external-image function for more details.

make-dither Function

Summary Makes a dither matrix of a given size.

Signature make-dither size => matrix

Arguments *size* An integer

Values *matrix* A dither matrix

Description The make-dither function makes a dither matrix of the

given size.

make-font-description

Function

Summary Returns a new font description object containing given font

attributes.

Signature make-font-description &rest font-attribute* => fdesc

Arguments font-attribute A font attribute

Values *fdesc* A font description object

Description The make-font-description function returns a new font

description object containing the given font attributes. There

is no error checking of the attributes at this point.

make-graphics-state

Function

Summary Creates a graphics state object.

Signature make-graphics-state =>

Arguments None

Values None

Description The make-graphics-state function creates a graphics state

object. Each graphics port has a graphics state associated with it, but you may want to create your own individual graphics states for use in specialized drawing operations. Graphics state objects do not consume local resources beyond dynamic memory for the structure (so you can be relaxed about creating them in some number if you really need to). Such objects are used in the with-graphics-state macro described below and modified using the following functions:

```
graphics-state-transform
graphics-state-foreground
graphics-state-background
graphics-state-operation
graphics-state-stipple
graphics-state-pattern
graphics-state-thickness
graphics-state-scale-thickness
graphics-state-dashed
graphics-state-dash
graphics-state-fill-style
graphics-state-line-end-style
graphics-state-line-joint-style
graphics-state-mask
graphics-state-mask-x
graphics-state-mask-y
graphics-state-font
```

These are the read and write (via setf) accessors for the graphics state slots. See the *LispWorks User Guide* for valid values for these accessors.

make-image-from-port

Function

Summary Makes an image out of a specified rectangle of a graphics port's contents. Signature make-image-from-port port & optional x y width height => image **Arguments** A graphics port port An integer X y An integer width An integer height An integer Values image An image

Description

The make-image-from-port function makes an image out of the specified rectangle of the port's contents. The default is the whole port, but a region can be specified using x, y, width, and height. The default value of x and y is 0.

make-transform Function

Summary Returns a new transform object initialized according to a set

of optional arguments.

Signature make-transform & optional $a \ b \ c \ d \ e \ f \Rightarrow transform$

Arguments a A real number

b A real number

c A real number

d A real number

e A real number

f A real number

Values transform A transform

Description

The make-transform function returns a new transform object initialized according to the optional args. The default args make the unit transform.

Default values are as follows: a and d are 1; b, c, e, and f are 0. The transform matrix is

a b (

c d 0

e f 1

for generalized two dimensional points of the form $(x\ y\ 1)$.

merge-font-descriptions

Function

Summary Returns a font description containing the attributes of two

specified font descriptions.

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 contents of *fdesc1* and *fdesc2* are not modi-

fied.

offset-rectangle

Function (inline)

Summary Offsets a rectangle by a given distance.

Signature offset-rectangle rectangle dx dy =>

Arguments rectangle A list of integers

dx A real numberdy A real number

Description The offset-rectangle function offsets the rectangle by the

distance (dx dy).

ordered-rectangle-union

Function

Summary Returns the union of two rectangles.

Signature	ordered-rectangle-union left-1 top-1 right-1 bottom-1
_	left-2 top-2 right-2 bottom-2 =>
	left ton right hottom

		ieit to
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
		A 1 1

top A real number
right A real number

bottom A real number

 $\hbox{\it Description}\qquad \hbox{\it The ordered-rectangle-union function returns four values:}$

the *left*, *top*, *right* and *bottom* of the union of the two rectangles specified in the arguments. The caller guarantees that each input rectangle is ordered, that is, the left values must be smaller or equal to the right values, and the top values must

be greater than or equal to the bottom ones.

See also rectangle-union

pixblt Function

Summary Copies one area of a graphics port to another area of a different graphics port.

Signature pixblt to-port operation from-port to-x to-y width height from-x from-y &optional depth =>

Arguments *to-port* A graphics port

operation A graphics state operation

An integer

from-port A graphics port

to-x An integer
to-y An integer
width An integer
height An integer
from-x An integer
from-y An integer

Description

depth

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 *depth* argument is available only to further optimize performance. If it is supplied, it should give the depth in pixels of both ports. Its presence obviates another call to port-depth in order to find out the destination port depth. The graphics port transforms are not used.

pixmap-port Class

Summary The class of pixmap graphics port objects.

Description The pixmap-port class is the class of pixmap graphics port

objects which can be used for drawing operations.

port-depth Function

Summary Returns the pixel depth of a port.

Signature port-depth port => result

Arguments *port* A graphics port

Values result An integer

Description The port-depth function returns the pixel depth of *port*.

port-height Function

Summary Returns the pixel height of a port.

Signature port-height port => result

Arguments port A graphics port

Values result An integer

Description The port-height function returns the pixel height of *port*.

port-string-height

Function

Summary Returns the height of a string drawn to a given port in pixels.

Signature port-string-height port string => height

Arguments port A graphics port

string A string

Values *height* An integer

Description The port-string-height function returns the height in pixels

of string when drawn to port. The font used is the one cur-

rently in the port's graphics state.

port-string-width

Function

Summary Returns the width of a string drawn to a given port in pixels.

Signature port-string-width port string => width

Arguments port A graphics port

string A string

Values width An integer

Description The port-string-width function returns the *width* in pixels

of string when drawn to port. The font used is the one cur-

rently in the port's graphics state.

port-width Function

Summary Returns the pixel width of a port

Signature port-width port => width

Arguments port A graphics port

Values width An integer

Description The port-width function returns the pixel width of *port*.

postmultiply-transforms

Function

Summary Postmultiplies two transforms.

Signature postmultiply-transforms transform1 transform2 =>

Arguments transform1 A transform

transfrom2 A transform

Description The postmultiply-transforms function postmultiplies the

partial 3 x 3 matrix represented by *transform1* by the partial 3 x 3 matrix represented by *transform2*, storing the result in *transform1*. In the result, the translation, scaling and rotation operations contained in *transform2* are effectively performed

after those in transform1.

transform1 = transform1 . transform2

premultiply-transforms

Function

Summary Premultiplies two transforms.

Signature premultiply-transforms transform1 transform2 =>

Arguments transform1 A transform

transform2 A transform

Description The premultiply-transforms function premultiplies the

partial 3 x 3 matrix represented by *transform1* by the partial 3 x 3 matrix represented by *transform2*, storing the result in *transform1*. In the result, the translation, scaling and rotation operations contained in *transform2* are effectively performed

before those in transform1.

transform1 = transform2 . transform1

read-and-convert-external-image

Function

Summary Returns an image converted from an external image read

from a file.

Signature read-and-convert-external-image gp file => image

Arguments gp A graphics port

file A pathname designator

Values image An image

Description Returns an image converted from an external image read

from file. The external image is returned as a second value.

read-external-image

Function

Summary Returns an external image read from a file.

Signature read-external-image file => image

Arguments file A pathname designator

Values image An external image

Description The read-external-image function returns an external

image read from file.

rectangle-bind Macro

Summary Binds four variables to the corners of a rectangle across a

body of code.

Signature rectangle-bind ((a b c d) rectangle) &body body => result

Arguments a A variable

b A variable

c A variable

d A variable

rectangles A rectangle

body A body of code

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 Returns the bottom element of a rectangle. Can be used to set

this element using setf.

Signature rectangle = bottom

Arguments rectangle A rectangle

Values bottom A real number

Description Returns and via setf sets the bottom element of rectangle.

rectangle-height

Macro

Summary Returns the height of a rectangle.

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-left Macro

Summary Returns the left element of a rectangle. Can be used to set this

element using setf.

Signature rectangle-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-right Macro

Summary Returns the right element of a rectangle. Can be used to set

this element using setf.

Signature rectangle-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-top Macro

Summary Returns the top element of a rectangle. Can be used to set this

element using setf.

Signature rectangle-top rectangle => top

Arguments rectangle A rectangle

Values top A real number

Description The rectangle-top macro returns and via setf sets the top

element of rectangle.

rectangle-union

Function

Summary Returns the four value of a union of two rectangles.

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.

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.

rect-bind Macro

Summary Binds four variables to the elements of a rectangle across a

body of code.

Signature rect-bind ((xy 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-bond macro binds x y width height to the appropri-

ate 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 load-

ing function.

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 identi-

fiers

image-load-function

A function

image-translation-table

An image translation table

Values

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.

register-image-translation

Function

Summary Registers an image identifier and image loading function

with a translation in an image translation table.

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

Values

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 call 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*.

reset-image-translation-table

Function

Summary Clears the image translation table hash tables.

Signature reset-image-translation-table &key image-translation-table

=>

Arguments image-translation-table

An image translation table

Values

Description The reset-image-translation-table function clears the

image translation table hash tables and set the default imageload-fn to read-and-convert-external-image. The default value of *image-translation-table* is *default-image-transla-

tion-table*.

separation Function

Summary Returns the distance between two points.

Signature separation $x1 \ y1 \ x2 \ y2 \Rightarrow dist$

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.

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

Values

Description The set-default-image-load-function function sets the

default image load function of *image-translation-table*. The default image load function is gp: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.

Signature set-graphics-port-coordinates port &key left top right

bottom =>

Arguments *port* A graphics port

left A real number

top A real number

right A real number

bottom A real number

Description The set-graphics-port-coordinates function modifies the

transform of the port is permanently such that the edges of

the port correspond to the values of the arguments.

Example The following code

changes the coordinates of the port so that the point (0 0) is in the exact center of the port and the edges are a unit distance away, with a right-handed coordinate system.

By default, *left* and *top* are 1.

set-graphics-state

Function

Summary Directly alters the graphics state of a graphics port according

to the keyword arguments supplied.

Signature set-graphics-state port &rest args

&key transform foreground background operation stipple pattern thickness scale-thickness

 $\ dashed\ dash\ line-end-style\ line-joint-style$

mask mask-x mask-y font

=>

Arguments *port* A graphics port

args

Values

Description This directly alters the graphics state of *port* according to the

keyword arg values. Unspecified args leave the associated

slots unchanged.

transform Type

Summary The transform type, defined for transform objects.

Description The transform type is the type defined for transform objects,

which are six-element lists of numbers.

transform-area Function

Summary Transforms a set of points and returns the resulting rectangle.

Signature transform-area transform x y width height => rectangle

Arguments transform A transform

x A real number

y A real number

width A real number

height A real number

Values rectangle A rectangle

Description The transform-area function transforms the points (x y) and

(x+width y+height) and returns the transformed rectangle as

(x y width height) values.

transform-distance

Function

Summary Transforms a distance vector by the rotation and scale of a

transform.

Signature transform-distance transform $dx dy => dx^2 dy^2$

Arguments transform A transform

dx A real number

dy A real number

Values dx^2 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 transla-

tion 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.

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 alter-

nating (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.

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.

Signature transform-point transform $x \ y \Rightarrow xnew \ ynew$

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.

Signature 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 alternat-

ing (*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 Function

Summary Returns the transform of two points representing the top-left

and bottom-right of a rectangle.

Signature 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 repre-

sented by the two points (*left top*) and (*right bottom*) by *trans-*

form.

undefine-font-alias Function

Summary Removes a font alias.

Signature undefine-font-alias keyword =>

Arguments keyword A keyword

Values

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 rect-

angle.

Signature union-rectangle rectangle left top right bottom => rectangle

Arguments rectangle A rectangle

left A real number

right A real number

top A real number

bottom A real number

Values rectangle A rectangle

Description The union-rectangle macro modifies the rectangle to be the

union of rectangle and (left top right bottom).

unit-transform-p

Function

Summary Returns t if a given transform is a unit transform.

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 trans-

form.

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.

Signature unless-empty-rect-bind ((x y width height) rectangle)

&body body => result

Arguments x A variable

y A variablewidth 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

 \emph{height} to the appropriate values from $\emph{rectangle}$ and if the \emph{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.

Signature untransform-distance transform $dx dy \Rightarrow x y$

Arguments transform A transform

dx A real number

dy A real number

Values x A real number

y A real number

Description The untransform-distance function transform the distance

(dx dy) by the rotation and scale of the effective inverse of *transform*. The translation in the inverse transform is ignored. The transformed distance (dx dy) is returned as two values.

untransform-distances

Function

Summary Transforms a list of integer pairs representing distances by

the inverse of a transform.

Signature untransform-distances transform distances => result

Arguments transform A transform

distances A list of pairs of real numbers

Values result A list of pairs of real numbers

Description The untransform-distances function transforms a list of

alternating (*dx dy*) pairs in *distances* by the effective inverse of *transform*. Transformed values are returned as a new list.

untransform-point

Function

Summary Transforms a point by multiplying it by the inverse of a given

transform.

Signature untransform-point transform $x \ y \Rightarrow x2 \ y2$

Arguments transform A transform

x A real number

y A real number

Values x2 A real number

y2 A real number

Description The untransform-point function transform the point (x y) by

effectively multiplying it by the inverse of *transform*. The

transformed (x y) is returned as two values.

untransform-points

Function

Summary Transforms a list of points by the inverse of a given trans-

form.

Signature untransform-points transform points & optional into => result

Arguments transform A transform

points A list of pairs of real numbers

into A list

Values result A list of pairs of real numbers

Description The untransform-points function transforms a list of alter-

nating (*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.

with-dither Macro

Summary Specifies a dither for use within a specified body of code.

Signature with-dither (dither-or-size) &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.

with-graphics-mask

Macro

Summary Binds the mask slot of a graphics port to a rectangular area

across the execution of a body of code.

Signature with-graphics-mask (port mask mask-x mask-y &key)

&body body => result

Arguments *port* A graphics port

mask A list of the form (x y width height) or nil

mask-x An integer

mask-y An integer

body A body of Lisp code

Values result The return value of the last form executed in

body

Description The with-graphics-mask macro binds the mask slot of port's

graphic state to a rectangular area across the execution of

body. By default, mask-x and mask-y are both 0.

with-graphics-rotation

Macro

Summary Performs a call to apply-rotation with a given angle for the

duration of the macro's body.

Signature with-graphics-rotation (port angle) &body body => result

Arguments *port* A graphics port

angle A real

body A body of Lisp code

Values result The return value of the last form executed in

body

Description The with-graphics-rotation marco performs a call to

(apply-rotation transform angle)

on the port's transform for the duration of the body of the

macro.

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.

Signature with-graphics-scale (port sx sy) &body body => result

Arguments A graphics port port

> A real number SXA real number SY

A body of Lisp code body

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

Binds the graphics state values of a port to a list of arguments Summary

and executes a body of code.

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) 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 keywords args are the same as the symbols of the slots in the graphics state, as described above. For example:

Arguments that are not supplied default to the current state of that slot in the graphics state.

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.

Signature with-graphics-transform (port transform) &body body

=> result

Arguments *port* A graphics port

transform A transform

body A body of Lisp code

Values result The return value of the last form executed in

body

Description The with-graphics-transform macro combines the trans-

form associated with the graphics port *port* with *transform* during the body of the macro. The port is given a new transform obtained by pre-multiplying its current transform with *transform*. This has the effect of *preceding* any translation, scaling and rotation operations specified in the body of the macro by those operations embodied in *transform*.

with-graphics-translation

Macro

Summary Applies a translation to a given port for the duration of the

macro.

Signature with-graphics-translation (port dx dy) &body body => result

Arguments *port* A graphics port

dx A real numberdy A real number

body A body of Lisp code

Values result The return value of the last form executed in

body

Description The with-graphics-translation macro performs a call to

(apply-translation transform dx dy)

on the port's transform for the duration of body of the macro.

with-inverse-graphics

Macro

Summary Executes all drawing function calls to a given port within the

body of the macro with foreground and background colors

swapped.

Signature with-inverse-graphics (port) &body => result

Arguments *port* A graphic port

body A body of Lisp code

Values result The return value of the last form executed in

body

Description The with-inverse-graphics macro ensures that all drawing

function calls to *port* within the body of the macro are executed with the foreground and background slots of the

graphics state of the port swapped around.

without-relative-drawing

Macro

Summary Evaluates a body of Lisp code with the *relative* and *collect*

internal variables of the port set to mil.

Signature without-relative-drawing (port) &body body => result

Arguments *port* A graphic port

body A body of Lisp code

Values result The return value of the last form executed in

body

Description The with-relative-drawing macro evaluates the code in

body with the relative and collect internal variables of the pixmap graphics port port set to nil to turn off the port's collecting of drawing bounds and automatic shifting of its origins. Use this macro only within a with-pixmap-graphics-port macro.

with-pixmap-graphics-port

Macro

Summary Binds a port to a new pixmap graphics port for the duration

of the macro's code body.

Signature with-pixmap-graphics-port (port pane width height

&key background collect relative clear)

&body body) => result

Arguments *port* A graphic port

pane An output pane

width An integer height An integer

background A color keyword

collect A boolean
relative A boolean
clear A list or t

body A body of Lisp code

Values result The return value of the last form executed in

body

Description The with-pixmap-graphics-port macro binds port to a new

pixmap graphics-port. *pane* and the other arguments are passed to create-pixmap-port. The *body* is then evaluated.

The port is destroyed when body returns.

with-transformed-area

Macro

Summary Transforms a rectangle using a port's transform, and binds

the resulting values to a variable across the evaluation of the

macro's body.

Signature with-transformed-area (points port left top right bottom)

&body body

Arguments *points* A variable

port A graphics port

left A real number

top A real number

right A real number

bottom A real number

body A body of Lisp code

Values result The return value of the last form executed in

body

Description The with-transformed-area macro transforms a rectangle,

binding the resulting four corner points to *points* for the duration of *body*. The *left top right bottom values* represent a rectangular area bounded by four points. The four points are transformed by the *port*'s transform and the list of eight values (alternating *x* and *y* values for four points) bound to the

points variable for the duration of the macro body.

with-transformed-point

Macro

Summary Binds a point transformed by a given ports transform to two

variables across the body of the macro.

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 portx A real numbery A real number

body A body of Lisp code

Values result The return value of the last form executed in

body

Description The with-transformed-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.

Signature with-transformed-points (points port) &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.	
Signature	with-transfor	med-rect (nx1 ny1 nx2 ny2 port x1 y1 x2 y2) &body body => result
Arguments	nx1	A variable
	ny1	A variable
	nx2	A variable
	ny2	A variable
	port	A graphics port
	<i>x</i> 1	A real number
	<i>y</i> 1	A real number
	x2	A real number
	<i>y</i> 2	A real number
	body	A body of Lisp code
Values	result	The return value of the last form executed in <i>body</i>
Description	During the evaluation of the with-transformed-rect macro <i>body</i> , the two points $(x1, y1)$ and $(x2, y2)$ are transformed by the port's current transform and the resulting values bound to the variables named by the $nx1 ny1 nx2 ny2$ args.	

write-external-image

Function

Summary Writes an external image to a file.

Signature write-external-image external-image file &key if-exists =>

Arguments eternal-image An external image

file A file

if-exists A keyword

Values

Description The write-external-image function writes an external

image to a file. By default, if-exists is :error. The if-exists argu-

ment is used in a call to open.

GP Reference Entries

Index

A	passing different variables 3
abort-dialog function 1	callbacks class 11
accessor functions	callbacks Slot 7
editor-pane-buffer 46	callback-type Sl0t 149, 152
activate-pane function 2	call-editor generic function 13
activep slot 106	capi-object class 13
adjust slot 26, 138	capi-object-property function 14
analyze-external-image function 174	caret-position slot 151
apply-rotation function 175	change-callback slot 152
apply-scale function 175	change-callback-type slot 152
apply-translation function 176	check-button class 15
attach-interface-for-callback	check-button-panel class 16
function 3	child slot 145
augment-font-description function 176	children-function slot 55 choice class 17
В	choice-selected-item generic function 19
background slot 144	choice-selected-items generic
beep-pane function 3	function 20
best-height slot 66	classes
best-width slot 66	button 3
best-x slot 66	button-panel 6
best-y slot 66	callbacks 11
buffer-name Slot 25, 45	capi-object 13
button class 3	check-button 15
button-panel class 6	check-button-panel 16
_	choice 17
C	collection 21
calculate-constraints generic	collector-pane 25
function 9	color-screen 26
calculate-layout generic function 10	column-layout 26
callback slot 4, 92, 152	display-pane 41
callback-data-function slot 93	drawn-pinboard-object 42
callbacks 11	editor-pane 44
for button panels 7	element 46
for buttons 5	external-image 195

form-layout 51	toolbar-object 163
graph-pane 55	x-y-adjustable-layout 171
grid-layout 58	clear-external-image-conversions
image 208	function 177
image-list 62	clear-graphics-port function 177
image-pinboard-object 63	clear-graphics-port-state
interactive-pane 64	function 178
interface 65	clear-rectangle function 178
item 71	collection class 21
item-pinboard-object 73	collection-search generic function 24
layout 74	collector-pane class 25
line-pinboard-object 75	color-screen class 26
listener-pane 76	column slot 58
list-panel 76	column-layout class 26
list-view 77	completion-function slot 152
menu 88	compress-external-image function 178
menu-component 90	compute-char-extents function 179
menu-item 91	confirm-change-function slot 152
menu-object 92	confirm-destroy-function slot 66
mono-screen 94	confirm-yes-or-no function 28
multiple-line-text-input-pane 95	contain function 28
option-pane 95	convert-external-image function 179
output-pane 96	convert-to-screen function 30
password-pane 103	copy-external-image function 180
pinboard-layout 104	copy-pixels function 180
pinboard-object 106	copy-transform function 181
pixmap-port 219	count-collection-items generic
progress-bar 118	function 30
push-button 127	create-pixmap-port function 182
push-button-panel 128	current-printer function 31
radio-button 131	
radio-button-panel 131	D
range-pane 132	data slot 71
right-angle-line-pinboard-	default slot 74
object 136	*default-image-translation-table*
row-layout 137	variable 173, 210
screen 139	define-command macro 31
simple-layout 143	define-font-alias function 183
simple-pane 143	define-interface macro 33
simple-pinboard-layout 145	define-layout macro 36
slider 147	define-menu macro 37
switchable-layout 147	depth slot 139
tab-layout 148	description slot 74
text-choice 151	description slot 148
text-input-pane 151	destroy generic function 38
titled-image-pinboard-object 154	destroy-callback slot 66
titled-menu-object 155	destroy-pixmap-port function 183
titled-pane 156	dialogs
title-pane 158	aborting 1
toolbar 159	
404	direction slot 75
toolbar-button 161	direction slot 75 disabled-image slot 4
toolbar-button 161 toolbar-component 162	

display-callback slot 42,96	focus
display-dialog function 39	moving to a new pane 2
display-message function 40	font slot 73, 144
display-message-for-pane function 41	font-description function 198
display-pane class 41	font-description-attributes
dither-color-spec function 184	function 198
draw-arc function 184	font-description-attribute-value
draw-arcs function 185	function 199
draw-character function 186	font-fixed-width-p function 199
draw-circle function 186	foreground slot 73, 144
draw-ellipse function 187	form-layout class 51
draw-image function 188	free-image function 200
draw-line function 189	functions
draw-lines function 189	abort-dialog 1
drawn-pinboard-object class 42	activate-pane 2
draw-pinboard-object generic	analyze-external-image 174
function 43	apply-rotation 175
draw-pinboard-object-highlighted	apply-scale 175
generic function 44	apply-translation 176
draw-pinboard-object-unhighlighted	attach-interface-for-callback 3
generic function 44	augment-font-description 176
draw-point function 190	beep-pane 3
draw-points function 190	capi-object-property 14
draw-polygon function 191	clear-external-image-
draw-polygons function 192	conversions 177
draw-rectangle function 193	clear-graphics-port 177
draw-rectangles function 193	clear-graphics-port-state 178
draw-string function 194	clear-rectangle 178
	compress-external-image 178
E	compute-char-extents 179
edge-pinboard-class slot 56	confirm-yes-or-no 28
editor-pane class 44	contain 28
editor-pane-buffer accessor function 46	convert-external-image 179
element class 46	convert-to-screen 30
enable slot 4	copy-external-image 180
enabled slot 45, 92, 95, 152	copy-pixels 180
enabled-function slot 92	copy-transform 181
enabled-slot slot 92	create-pixmap-port 182
ensure-interface-screen function 49	current-printer 31
exit-confirmer function 49	define-font-alias 183
exit-dialog function 50	destroy-pixmap-port 183
external-image class 195	display 39
external-image-color-table	display-dialog 39
function 195, 196	display-message 40
externalize-image function 196	display-message-for-pane 41
	dither-color-spec 184
F	draw-arc 184
find-best-font function 197	draw-arcs 185
find-matching-fonts function 198	draw-character 186
find-string-in-collection generic	draw-circle 186
function 51	draw-ellipse 187
	draw-image 188

draw-line 189	graphics-state-mask 215
draw-lines 189	graphics-state-mask-x 215
draw-point 190	graphics-state-mask-y 215
draw-points 190	graphics-state-operation 215
draw-polygon 191	graphics-state-pattern 215
draw-polygons 192	graphics-state-scale-thickness 215
draw-rectangle 193	graphics-state-stipple 215
draw-rectangles 193	graphics-state-thickness 215
draw-string 194	graphics-state-transform 215
ensure-interface-screen 49	hide-interface 61
exit-confirmer 49	image-freed-p 209
exit-dialog 50	image-loader 209
external-image-color-table 195,	image-set 64
196	image-translation 210
externalize-image 196	initialize-dithers 210
find-best-font 197	inset-rectangle 211
find-matching-fonts 198	inside-rectangle 211
font-description 198	invalidate-pane-constraints 70
font-description-attributes 198	invert-transform 212
font-description-attribute-	invoke-command 70
value 199	invoke-untranslated-command 71
font-fixed-width-p 199	load-image 212
free-image 200	lower-interface 83
get-bounds 200	make-dither 213
get-character-extent 201	make-font-description 214
get-char-ascent 202	make-general-image-set 84
get-char-descent 202	make-graphics-state 214
get-char-width 203	make-image-from-port 215
get-constraints 53	make-image-locator 85
get-enclosing-rectangle 203	make-resource-image-set 85
get-font-ascent 204	make-transform 216
get-font-average-width 204	map-typeout 87
get-font-descent 205	merge-font-descriptions 217
get-font-height 205	modify-editor-pane-buffer 94
get-font-width 205	offset-rectangle 217
get-graphics-state 206	ordered-rectangle-union 217
get-origin 206	page-setup-dialog 100
get-page-area 54	pixblt 218
get-printer-metrics 55	popup-confirmer 109
get-string-extent 207	port-depth 220
get-transform-scale 208	port-height 220
graphics-port-transform 208	port-string-height 220
graphics-state-background 215	port-string-width 221
graphics-state-dash 215	port-width 221
graphics-state-dashed 215	postmultiply-transforms 222
graphics-state-fill-style 215	premultiply-transforms 222
graphics-state-font 215	print-dialog 112
graphics-state-foreground 215	printer-metrics-device-height 114
graphics-state-line-end-	printer-metrics-device-width 115
style 215	printer-metrics-dpi-x 116
graphics-state-line-joint-	printer-metrics-dpi-y 116
style 215	printer-metrics-height 117
20,20 210	E WOOTTOD WOTENO II'

printer-metrics-width 117	G
print-file 113	gap slot 26, 138
print-text 114	generic functions
prompt-for-confirmation 118	calculate-constraints 9
prompt-for-file 119	calculate-layout 10
prompt-for-form 121	call-editor 13
prompt-for-forms 122	choice-selected-item 19
prompt-for-integer 123	choice-selected-items 20
prompt-for-string 124	collection-search 24
prompt-for-symbol 125	count-collection-items 30
prompt-with-list 126	destroy 38
quit-interface 129	draw-pinboard-object 43
raise-interface 133	draw-pinboard-object-
read-and-convert-external-	highlighted 44
image 223, 231	draw-pinboard-object-
read-external-image 223	unhighlighted 44
rectangle-union 226	find-string-in-collection 51
redisplay-menu-bar 134	get-collection-item 53
redraw-pinboard-layout 135	highlight-pinboard-object 61
redraw-pinboard-object 135	interpret-description 69
register-image-load-function 228	itemp 73
register-image-translation 229	make-container 83
remove-capi-object-property 136	map-collection-items 86
reset-image-translation-table 229	map-pane-children 86
separation 230	over-pinboard-object-p 99
set-default-image-load-	pane-adjusted-offset 100
function 231	pane-adjusted-position 101
set-geometric-hint 141	parse-layout-descriptor 103
set-graphics-port-coordinates 231	pinboard-object-at-position 107
set-graphics-state 232	pinboard-object-overlap-p 108
set-hint-table 141	print-collection-item 111
show-interface 142	redisplay-interface 134
simple-print-port 146	search-for-item 140
transform-area 233	set-button-panel-enabled 140
transform-distance 234	set-scroll-position 142
transform-distances 234	set-scroll-range 142
transform-is-rotated 235	top-level-interface 164
transform-point 235	top-level-interface-p 165
transform-points 235	unhighlight-pinboard 165
transform-rect 236	generic images 5
undefine-font-alias 237	get-bounds function 200
unit-transform-p 238	get-character-extent function 201
unmap-typeout 166	get-char-ascent function 202
untransform-distance 239	get-char-descent function 202
untransform-distances 239	get-char-width function 203
untransform-point 240	get-collection-item generic function 53
untransform-points 240	get-constraints function 53
update-pinboard-object 166	get-enclosing-rectangle function 203
update-toolbar 166	get-font-ascent function 204
write-external-image 251	get-font-average-width function 204
	get-font-descent function 205
	get-font-height function 205

get-font-width function 205 get-graphics-state function 206	image-pinboard-object class 63 images
get-origin function 206	generic 5
get-page-area function 54	image-set function 64 image-translation function 210
get-printer-metrics function 55 get-string-extent function 207	initialize-dithers function 210
get-string-extent function 207 get-transform-scale function 208	input-model slot 96
	inset-rectangle function 211
graphics-port-transform function 208	inside-rectangle function 211
graphics-state-background	interaction slot 4, 17
function 215	interaction styles 5
graphics-state-dash function 215	interactions
graphics-state-dashed function 215	for choice 18
graphics-state-fill-style	interactive-pane class 64
function 215	interface class 65
graphics-state-font function 215	interface Slot 46
graphics-state-foreground	interfaces Slot 139
function 215	interpret-description generic
graphics-state-line-end-style	function 69
function 215	invalidate-pane-constraints
graphics-state-line-joint-style	function 70
function 215	invert-transform function 212
graphics-state-mask function 215	invoke-command function 70
graphics-state-mask-x function 215	invoke-untranslated-command
graphics-state-mask-y function 215	function 71
graphics-state-operation	item class 71
function 215	item-map-function slot 22
graphics-state-pattern function 215	itemp generic function 73
graphics-state-scale-thickness	item-pinboard-object class 73
function 215	items slot 22, 88, 90, 148
graphics-state-stipple function 215	items-count-function slot 22
graphics-state-thickness	items-function slot 88, 90
function 215	items-get-function slot 22
graphics-state-transform	
function 215	K
graph-pane class 55	keep-selection-p slot 17
grid-layout class 58	key-function slot 148
TT	
Н	L
height slot 139	layout class 74
hide-interface function 61	layout slot 66
highlight-pinboard-object generic	layout-args slot 7
function 61	layout-class slot 7
horizontal-scroll slot 144	layout-function slot 55
	line-pinboard-object class 75
I	listener-pane class 76
image class 208	list-panel class 76
image slot 4	list-view class 77
image-freed-p function 209	load-image function 212
image-list class 62	*local-rgb-color-distance-red
image-loader function 209	weight* variable 197

lower-interface function 83	map-typeout function 87
	max-characters slot 152
M	max-height slot 47
macros	max-width slot 47
define-command 31	menu class 88
define-interface 33	menu-bar-items slot 66
define-layout 36	menu-component class 90
define-menu 37	menu-item class 91
rectangle-bind 223	menu-object class 92
rectangle-bottom 224	merge-font-descriptions function 217
rectangle-height 224	message slot 156
rectangle-left 225	message-args slot 157
rectangle-right 225	min-height slot 47
rectangle-top 226	min-width slot 46
rectangle-width 227	modify-editor-pane-buffer function 94
rect-bind 227	mono-screen class 94
undefine-menu 165	multiple-line-text-input-pane
union-rectangle 237	class 95
unless-empty-rect-bind 238	multiple-selection interaction style 5
with-atomic-redisplay 167	·
with-dither 241	N
with-document-pages 167	name slot 13
with-geometry 168	node-pane-function slot 56
with-graphics-mask 241	node-pinboard-class Slot 56
with-graphics-rotation 242	no-selection interaction style 5
with-graphics-scale 243	number slot 139
with-graphics-state 243	number stor 100
with-graphics-transform 244	0
with-graphics-translation 245	
with-inverse-graphics 246	offset-rectangle function 217
without-relative-drawing 246	option-pane class 95
with-page 169	ordered-rectangle-union function 217
with-page-transform 169	orientation slot 58
with-pixmap-graphics-port 247	output-pane class 96
with-print-job 170	output-pane Slot 106
4 0	over-pinboard-object-p generic
with-random-typeout 170 with-transformed-area 248	function 99
with-transformed-point 248	_
with-transformed-points 249	P
with-transformed-rect 250	page-setup-dialog function 100
	pane-adjusted-offset generic
make-container generic function 83 make-dither function 213	function 100
make-font-description function 214	pane-adjusted-position generic
make-general-image-set function 84	function 101
make-general-image-set function 04 make-graphics-state function 214	parent slot 46
make-image-from-port function 215	parse-layout-descriptor generic
make-image-locator function 85	function 103
make-resource-image-set function 85	password-pane class 103
make-transform function 216	pinboard-layout class 104
map-collection-items generic	pinboard-object class 106
function 86	pinboard-object-at-position generic
map-pane-children generic function 86	function 107

pinboard-object-overlap-p generic	read-and-convert-external-image
function 108	function 223, 231
pixblt function 218	read-external-image function 223
pixmap-port class 219	rectangle-bind macro 223
plist slot 14	rectangle-bottom macro 224
popup-callback slot 92	rectangle-height macro 224
popup-confirmer function 109	rectangle-left macro 225
port-depth function 220	rectangle-right macro 225
port-height function 220	rectangle-top macro 226
port-string-height function 220	rectangle-union function 226
port-string-width function 221	rectangle-width macro 227
port-width function 221	rect-bind macro 227
postmultiply-transforms	redisplay-interface generic
function 222	function 134
premultiply-transforms function 222	redisplay-menu-bar function 134
print-collection-item generic	redraw-pinboard-layout function 135
function 111	redraw-pinboard-object function 135
print-dialog function 112	register-image-load-function
printer-metrics-device-height	function 228
function 114	register-image-translation
printer-metrics-device-width	function 229
function 115	remove-capi-object-property
printer-metrics-dpi-x function 116	function 136
printer-metrics-dpi-y function 116	reset-image-translation-table
printer-metrics-height function 117	function 229
printer-metrics-width function 117	right-angle-line-pinboard-object
print-file function 113	class 136
print-function slot 22, 71, 149	roots slot 55
print-text function 114	row-layout class 137
progress-bar class 118	rows slot 58
prompt-for-confirmation	
function 118	S
prompt-for-file function 119	screen class 139
prompt-for-form function 121	scroll-callback slot 96
prompt-for-forms function 122	search-for-item generic function 140
prompt-for-integer function 123	selected slot 4, $7\overline{2}$
prompt-for-string function 124	selected-item slot 17
prompt-for-symbol function 125	selected-items slot 17
prompt-with-list function 126	selection slot 17
push-button class 127	selection-callback slot 149
push-button-panel class 128	separation function 230
0	set-button-panel-enabled-items
Q	generic function 140
quit-interface function 129	set-default-image-load-function function 231
R	set-geometric-hint function 141
radio-button class 131	set-graphics-port-coordinates
radio-button-panel class 131	function 231
raise-interface function 133	set-graphics-state function 232
range-pane class 132	set-hint-table function 141
ratios slot 26, 137	set-scroll-position generic
	function 142

set-scroll-range generic function 142 U show-interface function 142 undefine-font-alias function 237 simple-layout class 143 undefine-menu macro 165 simple-pane class 143 unhighlight-pinboard-object generic simple-pinboard-layout class 145 function 165 simple-print-port function 146 uniform-size-p slot 26, 138 single-selection interaction style 5 union-rectangle macro 237 slider class 147 *unit-transform* special variable 174 special variables unit-transform-p function 238 *unit-transform* 174 unless-empty-rect-bind macro 238 stream slot 25.65 unmap-typeout function 166 streams 25 untransform-distance function 239 switchable-layout class 147 untransform-distances function 239 untransform-point function 240 Т untransform-points function 240 tab-layout class 148 update-pinboard-object function 166 test-function slot 22 update-toolbar function 166 text slot 41, 45, 71, 151, 158 text-choice class 151 text-input-pane class 151 variables title slot 66, 154, 155, 156 *default-image-translationtitle-adjust slot 52, 156 table* 173, 210 title-args slot 156 *local-rgb-color-distance-redtitled-image-pinboard-object weight* 197 class 154 vertical-adjustment slot 52 titled-menu-object class 155 vertical-gap slot 52 titled-pane class 156 vertical-scroll slot 144 title-function slot 155 visible-border slot 144 title-gap slot 52 visible-child slot 147 title-pane class 158 visible-child-function slot 148 title-position slot 156 toolbar class 159 toolbar-button class 161 width slot 139 toolbar-component class 162 with-atomic-redisplay macro 167 toolbar-object class 163 with-dither macro 241 top-level-function slot 65 with-document-pages macro 167 top-level-interface generic with-geometry macro 168 function 164 with-graphics-mask macro 241 top-level-interface-p generic with-graphics-rotation macro 242 function 165 with-graphics-scale macro 243 transform type 233 with-graphics-state macro 243 transform-area function 233 with-graphics-transform macro 244 transform-distance function 234 with-graphics-translation macro 245 transform-distances function 234 with-inverse-graphics macro 246 transform-is-rotated function 235 without-relative-drawing macro 246 transform-point function 235 with-page macro 169 transform-points function 235 with-page-transform macro 169 transform-rect function 236 with-pixmap-graphics-port macro 247 type slot 136 with-print-job macro 170 types with-random-typeout macro 170 transform 233 with-transformed-area macro 248

with-transformed-point macro 248 with-transformed-points macro 249 with-transformed-rect macro 250 write-external-image function 251

\mathbf{X}

x slot 47
x-adjust slot 154,171
x-gap slot 58
x-ratios slot 58
x-uniform-size-p slot 58
x-y-adjustable-layout class 171

Y

y slot 47 y-adjust slot 171 y-gap slot 58 y-ratios slot 58 y-uniform-size-p slot 59