Documentation Library

Gamma/Photon Reference Volume 1: Functions

Using Gamma™ with the Photon microGUI®, Version 1.1

Cogent Real-Time Systems, Inc.

August 11, 2010

Gamma/Photon Reference Volume 1: Functions: Using Gamma™ with the Photon microGUI®, Version 1.1

Gamma is the only dynamic language currently available for developing GUIs in the Photon environment. Gamma enhances Photon and PhAB with easy-to-use callbacks, expanded functionality for reusing widgets, and an object-oriented syntax.

Published August 11, 2010
Cogent Real-Time Systems, Inc.
162 Guelph Street, Suite 253
Georgetown, Ontario
Canada, L7G 5X7

Toll Free: 1 (888) 628-2028
Tel: 1 (905) 702-7851
Fax: 1 (905) 702-7850

Information Email: info@cogent.ca
Tech Support Email: support@cogent.ca
Web Site: www.cogent.ca

Revision History

Revision 3.4-1	August 2004 Compatible with Cascade DataHub and Cascade Connect Version 5.0.
Revision 3.3-1	September 2001 Source code compatible across QNX 4 and QNX 6.
Revision 3.2-1	September 2000 Renamed "Gamma/Photon", changed function syntax.
Revision 1.1	March 2000 Edited Widget Classes, expanded Programmer’s Manual.
Revision beta 1.0	February 2000 Expanded existing function references, added class references and Programmer’s Manual.

Copyright, trademark, and software license information.

Copyright Notice

Trademark Notice

END-USER LICENSE AGREEMENT FOR COGENT SOFTWARE

IMPORTANT - READ CAREFULLY: This End-User License Agreement ("EULA") is a legal agreement between you (either an individual or a single entity) and Cogent Real-Time Systems Inc. ("Cogent") of 162 Guelph Street, Suite 253, Georgetown, Ontario, L7G 5X7, Canada (Tel: 905-702-7851, Fax: 905-702-7850), from whom you acquired the Cogent software product(s) ("SOFTWARE PRODUCT" or "SOFTWARE"), either directly from Cogent or through one of Cogent's authorized resellers.
The SOFTWARE PRODUCT includes computer software, any associated media, any printed materials, and any "online" or electronic documentation. By installing, copying or otherwise using the SOFTWARE PRODUCT, you agree to be bound by the terms of this EULA. If you do not agree with the terms of this EULA, Cogent is unwilling to license the SOFTWARE PRODUCT to you. In such event, you may not use or copy the SOFTWARE PRODUCT, and you should promptly contact Cogent for instructions on return of the unused product(s) for a refund.

SOFTWARE PRODUCT LICENSE

The SOFTWARE PRODUCT is protected by copyright laws and copyright treaties, as well as other intellectual property laws and treaties. The SOFTWARE PRODUCT is licensed, not sold.

1. **EVALUATION USE:** This software is distributed as "Free for Evaluation", and with a per-use royalty for Commercial Use, where "Free for Evaluation" means to evaluate Cogent's software and to do exploratory development and "proof of concept" prototyping of software applications, and where "Free for Evaluation" specifically excludes without limitation:
    i. use of the SOFTWARE PRODUCT in a business setting or in support of a business activity,
    ii. development of a system to be used for commercial gain, whether to be sold or to be used within a company, partnership, organization or entity that transacts commercial business,
    iii. the use of the SOFTWARE PRODUCT in a commercial business for any reason other than exploratory development and "proof of concept" prototyping, even if the SOFTWARE PRODUCT is not incorporated into an application or product to be sold,
    iv. the use of the SOFTWARE PRODUCT to enable the use of another application that was developed with the SOFTWARE PRODUCT,
    v. inclusion of the SOFTWARE PRODUCT in a collection of software, whether that collection is sold, given away, or made part of a larger collection.
    vi. inclusion of the SOFTWARE PRODUCT in another product, whether or not that other product is sold, given away, or made part of a larger product.

2. **COMMERCIAL USE:** COMMERCIAL USE is any use that is not specifically defined in this license as EVALUATION USE.

3. **GRANT OF LICENSE:** This EULA covers both COMMERCIAL and EVALUATION USE of the SOFTWARE PRODUCT. Either clause (A) or (B) of this section will apply to you, depending on your actual use of the SOFTWARE PRODUCT. If you have not purchased a license of the SOFTWARE PRODUCT from Cogent or one of Cogent's authorized resellers, then you may not use the product for COMMERCIAL USE.
    A. **GRANT OF LICENSE (EVALUATION USE):** This EULA grants you the following non-exclusive rights when used for EVALUATION purposes:

Software: You may use the SOFTWARE PRODUCT on any number of computers, either stand-alone, or on a network, so long as every use of the SOFTWARE PRODUCT is for EVALUATION USE. You may reproduce the SOFTWARE PRODUCT, but only as reasonably required to install and use it in accordance with this LICENSE or to follow your normal back-up practices.

Subject to the license expressly granted above, you obtain no right, title or interest in or to the SOFTWARE PRODUCT or related documentation, including but not limited to any copyright, patent, trade secret or other proprietary rights therein. All whole or partial copies of the SOFTWARE PRODUCT remain property of Cogent and will be considered part of the SOFTWARE PRODUCT for the purpose of this EULA.

Unless expressly permitted under this EULA or otherwise by Cogent, you will not:

i. use, reproduce, modify, adapt, translate or otherwise transmit the SOFTWARE PRODUCT or related components, in whole or in part;

ii. rent, lease, license, transfer or otherwise provide access to the SOFTWARE PRODUCT or related components;

iii. alter, remove or cover proprietary notices in or on the SOFTWARE PRODUCT, related documentation or storage media;

iv. export the SOFTWARE PRODUCT from the country in which it was provided to you by Cogent or its authorized reseller;

v. use a multi-processor version of the SOFTWARE PRODUCT in a network larger than that for which you have paid the corresponding multi-processor fees;

vi. decompile, disassemble or otherwise attempt or assist others to reverse engineer the SOFTWARE PRODUCT;

vii. circumvent, disable or otherwise render ineffective any demonstration time-outs, locks on functionality or any other restrictions on use in the SOFTWARE PRODUCT;

viii. circumvent, disable or otherwise render ineffective any license verification mechanisms used by the SOFTWARE PRODUCT;

ix. use the SOFTWARE PRODUCT in any application that is intended to create or could, in the event of malfunction or failure, cause personal injury or property damage; or

x. make use of the SOFTWARE PRODUCT for commercial gain, whether directly, indirectly or incidentally.

B. **GRANT OF LICENSE (COMMERCIAL USE):** This EULA grants you the following non-exclusive rights when used for COMMERCIAL purposes:

Software: You may use the SOFTWARE PRODUCT on one computer, or if the SOFTWARE PRODUCT is a multi-processor version - on one node of a network, either: (i) as a development systems for the purpose of creating value-added software applications in accordance with related Cogent documentation; or (ii) as a single run-time copy for use as an integral part of such an application. This includes reproduction and configuration of the SOFTWARE PRODUCT, but only as reasonably required to install and use it in association with your licensed processor or to follow your normal back-up practices.

Storage/Network Use: You may also store or install a copy of the SOFTWARE PRODUCT on one computer to allow your other computers to use the SOFTWARE PRODUCT over an internal network, and distribute the SOFTWARE PRODUCT to your other computers over an internal network. However, you must acquire and dedicate a license for the SOFTWARE PRODUCT for each computer on which the SOFTWARE PRODUCT is used or to which it is distributed. A license for the SOFTWARE PRODUCT may not be shared or used concurrently on different computers.

Subject to the license expressly granted above, you obtain no right, title or interest in or to the SOFTWARE PRODUCT or related documentation, including but not limited to any copyright, patent, trade secret or other proprietary rights therein. All whole or partial copies of the SOFTWARE PRODUCT remain property of Cogent and will be considered part of the SOFTWARE PRODUCT for the purpose of this EULA.

Unless expressly permitted under this EULA or otherwise by Cogent, you will not:

i. use, reproduce, modify, adapt, translate or otherwise transmit the SOFTWARE PRODUCT or related components, in whole or in part;

ii. rent, lease, license, transfer or otherwise provide access to the SOFTWARE PRODUCT or related components;

iii. alter, remove or cover proprietary notices in or on the SOFTWARE PRODUCT, related documentation or storage media;

iv. export the SOFTWARE PRODUCT from the country in which it was provided to you by Cogent or its authorized reseller;

v. use a multi-processor version of the SOFTWARE PRODUCT in a network larger than that for which you have paid the corresponding multi-processor fees;

vi. decompile, disassemble or otherwise attempt or assist others to reverse engineer the SOFTWARE PRODUCT;

vii. circumvent, disable or otherwise render ineffective any demonstration time-outs, locks on functionality or any other restrictions on use in the SOFTWARE PRODUCT;

viii. circumvent, disable or otherwise render ineffective any license verification mechanisms used by the SOFTWARE PRODUCT, or

ix. use the SOFTWARE PRODUCT in any application that is intended to create or could, in the event of malfunction or failure, cause personal injury or property damage.

4. **WARRANTY:** Cogent cannot warrant that the SOFTWARE PRODUCT will function in accordance with related documentation in every combination of hardware platform, software environment and SOFTWARE PRODUCT configuration. You acknowledge that software bugs are likely to be identified when the SOFTWARE PRODUCT is used in your particular application. You therefore accept the responsibility of satisfying yourself that the SOFTWARE PRODUCT is suitable for your intended use. This includes conducting exhaustive testing of your application prior to its initial release and prior to the release of any related hardware or software modifications or enhancements.

   Subject to documentation errors, Cogent warrants to you for a period of ninety (90) days from acceptance of this EULA (as provided above) that the SOFTWARE PRODUCT as delivered by Cogent is capable of performing the functions described in related Cogent user documentation when used on appropriate hardware. Cogent also warrants that any enclosed disk(s) will be free from defects in material and workmanship under normal use for a period of ninety (90) days from acceptance of this EULA. Cogent is not responsible for disk defects that result from accident or abuse. Your sole remedy for any breach of warranty will be either: i) terminate this EULA and receive a refund of any amount paid to Cogent for the SOFTWARE PRODUCT, or ii) to receive a replacement disk.

5. **LIMITATIONS:** Except as expressly warranted above, the SOFTWARE PRODUCT, any related documentation and disks are provided "as is" without other warranties or conditions of any kind, including but not limited to implied warranties of merchantability, fitness for a particular purpose and non-infringement. You assume the entire risk as to the results and performance of the SOFTWARE PRODUCT. Nothing stated in this EULA will imply that the operation of the SOFTWARE PRODUCT will be uninterrupted or error free or that any errors will be corrected. Other written or oral statements by Cogent, its representatives or others do not constitute warranties or conditions of Cogent.

   In no event will Cogent (or its officers, employees, suppliers, distributors, or licensors: collectively "Its Representatives") be liable to you for any indirect, incidental, special or consequential damages whatsoever, including but not limited to loss of revenue, lost or damaged data or other commercial or economic loss, arising out of any breach of this EULA, any use or inability to use the SOFTWARE PRODUCT or any claim made by a third party, even if Cogent (or Its Representatives) have been advised of the possibility of such damage or claim. In no event will the aggregate liability of Cogent (or that of Its Representatives) for any damages or claim, whether in contract, tort or otherwise, exceed the amount paid by you for the SOFTWARE PRODUCT.

   These limitations shall apply whether or not the alleged breach or default is a breach of a fundamental condition or term, or a fundamental breach. Some jurisdictions do not allow the exclusion or limitation of liability for consequential or incidental damages, or certain limitations of implied warranties. Therefore the above limitation may not apply to you.

6. **DESCRIPTION OF OTHER RIGHTS AND LIMITATIONS:**

   Separation of Components. The SOFTWARE PRODUCT is licensed as a single product. Its component parts may not be separated for use on more than one computer.

Termination. Without prejudice to any other rights, Cogent may terminate this EULA if you fail to comply with the terms and conditions of this EULA. In such an event, you must destroy all copies of the SOFTWARE PRODUCT and all of its component parts.

7. **UPGRADES:** If the SOFTWARE PRODUCT is an upgrade from another product, whether from Cogent or another supplier, you may use or transfer the SOFTWARE PRODUCT only in conjunction with that upgrade product, unless you destroy the upgraded product. If the SOFTWARE PRODUCT is an upgrade of a Cogent product, you now may use that upgraded product only in accordance with this EULA. If the SOFTWARE PRODUCT is an upgrade of a component of a package of software programs which you licensed as a single product, the SOFTWARE PRODUCT may be used and transferred only as part of that single product package and may not be separated for use on more than one computer.
8. **COPYRIGHT:** All title and copyrights in and to the SOFTWARE PRODUCT (including but not limited to any images, photographs, animations, video, audio, music, text and ’applets", incorporated into the SOFTWARE PRODUCT), any accompanying printed material, and any copies of the SOFTWARE PRODUCT, are owned by Cogent or its suppliers. You may not copy the printed materials accompanying the SOFTWARE PRODUCT. All rights not specifically granted under this EULA are reserved by Cogent.
9. **PRODUCT SUPPORT:** Cogent has no obligation under this EULA to provide maintenance, support or training.
10. **RESTRICTED RIGHTS:** Use, duplication, or disclosure by the U.S. Government is subject to restrictions as provided in DFARS 227.7202-1(a) and 227.7202-3(a) (1995), DFARS 252.227-7013(c)(1)(ii) (OCT 1988), FAR 12.212(a)(1995), FAR 52.227-19, or FAR 52.227-14 (ALT III), as appropriate. Manufacturer is Cogent Real-Time Systems Inc. 162 Guelph Street, Suite 253, Georgetown, Ontario, L7G 5X7, Canada.
11. **GOVERNING LAW:** This Software License Agreement is governed by the laws of the Province of Ontario, Canada. You irrevocably attorn to the jurisdiction of the courts of the Province of Ontario and agree to commence any litigation that may arise hereunder in the courts located in the Judicial District of Peel, Province of Ontario.

Table of Contents

List of Tables

Chapter 1. What is Gamma?

Gamma is an interpreter, a high-level programming language that has been designed and optimized to reduce the time required for building applications. It has support for the Photon GIU in QNX, and the GTK GUI in Linux and QNX 6. It also has extensions that support HTTP and MySQL.

With Gamma a user can quickly implement algorithms that are far harder to express in other languages such as C. Gamma lets the developer take advantage of many time-saving features such as memory management and improved GUI support. These features, coupled with the ability to fully interact with and debug programs as they run, mean that developers can build, test and refine applications in a shorter time frame than when using other development platforms.

Gamma programs are small, fast and reliable. Gamma is easily embedded into today's smart appliances and web devices.

Gamma is an improved and expanded version of our previous Slang Programming Language for QNX and Photon. Gamma is available on QNX 4, QNX 6 and Linux, and is being ported to Microsoft Windows.

The implementation of Gamma is based on a powerful SCADALisp engine. SCADALisp is a dialect of the Lisp programming language which has been optimized for performance and memory usage, and enhanced with a number of internal functions. All references in this manual to Lisp are in fact to the SCADALisp dialect of Lisp.

You could say Gamma's object language is Lisp, just like Assembler is the object language for C. Knowing Lisp is not a requirement for using Gamma, but it can be helpful. All necessary information on Lisp and how it relates to Gamma is in the Input and Output chapter of this guide.

Chapter 2. System Requirements

QNX 6

- QNX 6.1.0 or later.

QNX 4

- QNX 4.23A or later.
- (For Gamma/Photon) Photon 1.14 or later.

Linux

- Linux 2.4 or later.
- (For Gamma/GTK) GTK 1.2.8.
- The SRR IPC kernel module, which includes a synchronous message passing library modeled on the QNX 4 send/receive/reply message-passing API. This module installs automatically, but requires a C compiler for the installation. You can get more information and/or download this module at the Cogent Web Site.

  

  This module may not be necessary for some Gamma applications, but it is required for any use of timers, event handling, or inter-process communication.

I. PhAB Functions

Table of Contents

PhabAttachWidgets

PhabAttachWidgets — assigns widgets to a class as instance variables.

Syntax

PhabAttachWidgets (*class*, *filename*)

Arguments

filename

The name of a widget file, including a pathname if necessary.

class

The name of a class.

Returns

A definition for the .PhabInstantiate method of the *class*, in Lisp syntax.

Description

This function reads a widget file create in PhAB and constructs a template that can be used to create a window. Any named widget becomes an instance variable of the *class*, and can be accessed as such. This function is similar to PhabLoad, but it provides more flexibility, since it creates globally-defined class with instance variables corresponding to each widget, rather than having each widget be defined by a global variable. It can also be used to create multiple instances of a group of widgets within a single window, as explained below.

The class gets created with three class variables: PhabInstantiate (holds the .PhabInstantiate method explained below), _phab_template (location and other information on widgets in the widget hierarchy), and _phab_template_file (path and file name of the widget file).

There are three steps involved in using this function:

1. Create a class that the widgets will be assigned to.
2. Call the PhabAttachWidgets function with that class as a parameter.
3. Instantiate the widgets using the .PhabInstantiate method that gets created by the function.

The .PhabInstantiate method

Syntax

.PhabInstantiate(*use_root*)

Arguments

use_root A flag that (set to t or nil) specifies whether the root widget (generally the main window) will be used.

Returns

The widget hierarchy on success, else nil.

Description

This method is required to instantiate the widgets attached by `PhabAttachWidgets`. You can call it at any time so long as you have an instance of a class that has been modified by `PhabAttachWidgets`. We suggest putting `.PhabInstantiate` in the constructor because it needs to be called even when constructing classes that are derived from your "Phab window" class.

Example

These examples, `ex_PhabAttachWidgets.g` and `ex_PhabAttachWidgets2.g`, are included in the product distribution.

This example, `ex_PhabAttachWidgets.g`, sets the *`use_root`* parameter of `.PhabInstantiate` to `t` (true), which causes the widgets to appear in their original window.

```
#!/usr/cogent/bin/phgamma

/* This example creates a class, attaches widgets from a Phab
 * file to the class, and sets up a constructor for the class that
 * instantiates the widgets.  Then it demonstrates how the widgets
 * are now accessible as instance variables of the class. */

PtInit(nil);
require_lisp("PhabTemplate.lsp");

/* Create the class.*/
class DemoWindow
{
  window;
}

/* Attach the widgets to the class. */
PhabAttachWidgets (DemoWindow, string(_os_, "-WidgetFiles/wgt/loadwindow.wgtw"));

/* Set up the constructor to instantiate the widgets inside the window. */
method DemoWindow.constructor ()
{
  .window = PhabRoot(.PhabInstantiate (t));
}

/* Create a new instance of the class and realize it. */
load_win = new (DemoWindow);
PtRealizeWidget (load_win.window);

/* Change the text string and assign a callback to a button
 * by calling the widgets as instances of the load_win class.*/
load_win.PtButtonTwo.text_string = "Exit";
PtAttachCallback(load_win.PtButtonTwo, Pt_CB_ACTIVATE, #exit_program(-1));

PtMainLoop();
```

This example, `ex_PhabAttachWidgets2.g`, sets the *`use_root`* parameter of `.PhabInstantiate` to `nil` (false), which gives you the freedom to put the widgets in any container, and/or to put multiple copies of them into the same containiner.

```
#!/usr/cogent/bin/phgamma

/* This example creates a class, attaches widgets from a Phab
 * file to the class, and sets up a constructor for the class that
 * instantiates the widgets.  Then it demonstrates how the widgets
 * are now accessible as instance variables of the class. */

PtInit(nil);
require_lisp("PhabTemplate.lsp");
require_lisp("PhotonWidgets.lsp");
```

```
/* Create the class.*/
class DemoWidgets
{
  widgets;
}

/* Attach the widgets to the class. */
PhabAttachWidgets (DemoWidgets, string(_os_, "-WidgetFiles/wgt/loadwindow.wgtw"));

/* Set up the constructor to instantiate the widgets, but not in a window. */
method DemoWidgets.constructor ()
{
  .widgets = PhabChildren(.PhabInstantiate (nil));
}

/* Create a window to hold instances of the class. */
window = new(PtWindow);
window.SetDim(500, 200);
window.fill_color = 0xaaccdd;

/* Create two instances of the class. */
dwidgets1 = new (DemoWidgets);

PtSetParentWidget(window);
dwidgets2 = new (DemoWidgets);
dwidgets2.MainPane.SetPos(260, 23);

/* Change the text string and assign a callback to a button
 * by calling the widgets as instances of the DemoWidgets class.*/
dwidgets1.PtButtonTwo.text_string = "Exit";
PtAttachCallback(dwidgets1.PtButtonTwo, Pt_CB_ACTIVATE, #exit_program(-1));

PtRealizeWidget (window);
PtMainLoop();
```

See Also

PhabLoad

PhabChildren

`PhabChildren` — a wrapper for the `cdr` function.

Syntax

`PhabChildren (`*`widget_hierarchy`*`)`

Arguments

`widget_hierarchy`

The name of an instantiated widget hierarchy, as returned from a call to `.PhabInstantiate`, which is a method of the class created by `PhabAttachWidgets`.

Returns

The children of a widget heirarchy, as a list of lists.

Description

This is a convenience function that provides a more meaningful name for the `cdr` function, when used in the context of a widget hierarchy. Since Gamma handles widget hierarchies as lists, the cdr of the list is a list that contains all of the children of the root widget. The children are actually each a single element of a list, so the return value becomes a list of lists, with each sub-list holding one child.

See Also

`PhabRoot`

PhabCreateWidgets

`PhabCreateWidgets` — creates widgets from existing definitions.

Syntax

`PhabCreateWidgets (`*`widget_defs`*`, `*`parent`*`, `*`name_prefix`*`)`

Arguments

`widget_defs`

A list of widget definitions most likely generated using the library functions PhabLoad or PhabReadWidgetFile and its related functions.

`parent`

The parent for the new widgets. Pass `nil` for no parent.

`name_prefix`

A string prefix to use when creating names for newly created widgets. Pass `nil` to use existing widget names.

Returns

A list of widgets, or `nil`.

Description

This function is used when creating new windows from a window definition. Typically, but not necessarily, the window template is created using PhAB. New names are optionally given to each window widget.

Example

This example, `ex_PhabCreateWidgets.g`, is included in the product distribution.

```
#!/usr/cogent/bin/phgamma

/*
This example reads a widget file that contains 8 buttons with .gif
images on them.  It creates the widgets and puts them into two
windows.  The first window contains all of the widgets, while the
second window contains a widget of your choice, which you can enter as
the third argument on the command line. To run this example, you must
be in a directory that can access the .wgt file.

The example takes an argument, <button name>, which can be one of:
Go, Stop, Pause, Done, Increase, Decrease, Left or Right.
*/

arg = string(cadr(argv));

if(string_p(arg))
{
  if((strcmp(arg, "Go") == 0) ||
  (strcmp(arg, "Stop") == 0) ||
  (strcmp(arg, "Pause") == 0) ||
  (strcmp(arg, "Done") == 0) ||
  (strcmp(arg, "Increase") == 0) ||
  (strcmp(arg, "Decrease") == 0) ||
```

```
 (strcmp(arg, "Left") == 0) ||
 (strcmp(arg, "Right") == 0))
 {

PtInit(nil);

/*
 * Rename some car/cdr functions to facilitate understanding.
 */
PhabGetTop = car;
PhabGetRoot = car;
PhabGetChildren = cdr;
PhabGetLevel = car;
PhabGetType = cadr;
PhabGetName = caddr;

require_lisp("PhotonWidgets.lsp");
require_lisp("PhabTemplate.lsp");

/*
 * Read the widget definitions, and access them.
 */
file = string(_os_, "-WidgetFiles/wgt/buttons.wgtw");
defs = PhabReadWidgetFile(file);
tree = PhabGetTop(defs);
children = PhabGetChildren(tree);

/*
 * Make a function to index the buttons by number.
 */
function index_buttons(buttons)
  {
 local index = 1;
 indexlist = list();

 with button in buttons do
 {
   local name = PhabGetName(car(button));
   local name = PhabGetName(car(button));
   indexlist = cons(list(name, index), indexlist);
   index ++;
 }
 indexlist;
  }

/*
 * Make a function to look up a requested button.
 */
function lookup (name, list)
  {
 car(assoc(name, list));
  }

/*
 * Make a new window, create all the buttons, and display them
 * in the window.
 */
win = new(PtWindow);
win.SetArea(100,50,120,500);
win.title = "All";
for (i=0;i<8;i++)
{
  eachbut = car(car(PhabCreateWidgets(list(car(nth_cdr(children,i)))
           ,nil,nil)));
  eachbut.SetPos(25, (i * 60));
  PtRealizeWidget(eachbut);
  PtExtentWidget(win);
}
```

```
 /*
  * Find the requested button.
  */
 buttonlist = index_buttons(children);
 argument = cadr(argv);
 found = lookup(symbol(argument), buttonlist);
 index = cadr(found) - 1;

 /*
  * Make a new window, create the requested button, and display it.
  */
 win2 = new(PtWindow);
 win2.SetArea(250,50,100,140);
 win2.title = "Request";
 anybut = car(car(PhabCreateWidgets(list(car(nth_cdr(children,index)))
            ,nil,nil)));
 anybut.SetPos(20, 20);

 exitbut = new(PtButton);
 exitbut.SetArea(30, 100, 40, 25);
 exitbut.text_string = "Exit";
 PtAttachCallback(exitbut,Pt_CB_ACTIVATE,#exit_program(1));


 PtRealizeWidget(win);
 PtMainLoop();
  }
  else
 princ ("Please enter an argument, one of: Go Stop Pause Done Increase Decrease Left Right \n");
}
else
princ ("Please enter an argument, one of: Go Stop Pause Done Increase Decrease Left Right \n");
```

See Also

PhabReadWidgetFile

PhabLoad

`PhabLoad` — reads, creates, names and realizes a widget hierarchy.

Syntax

```
PhabLoad (filename, realize?)
```

Arguments

`filename`

The name of the widget file to load.

`realize`

`t` or `nil`. The default (`t`) realizes the top-level window widget before returning the window hierarchy.

Returns

The top-level window hierarchy on success, otherwise `nil`.

Description

This is a convenience function that creates and returns a window hierarchy from a filename by calling `PhabReadWidgetFile`, `PhabCreateWidgets`, and `PhabNameWidgets` in succession. And unless *`realize`* is set to `nil`, `PhabLoad` also calls `PtRealizeWidget` on the top-level window widget, before returning the window hierarchy.

Because the widgets created by this function are accessed as global variables, it is generally preferred to use the PhabAttachWidgets function.

To use this function, you must load the PhabTemplate.lsp library with a call to **`require_lisp("PhabTemplate.lsp")`**. The filename is searched for in the currrent `_require_path_` (see PhabReadWidgetFile).

Example

This example, `ex_PhabLoad.g`, is included in the product distribution.

```
#!/usr/cogent/bin/phgamma

/* This example loads and displays a window and all its children.*/

PtInit(nil);
require_lisp("PhabTemplate.lsp");

PhabLoad (string(_os_, "-WidgetFiles/wgt/loadwindow.wgtw"));


PtMainLoop();
```

See Also

PhabReadWidgetFile, PhabCreateWidgets, PhabNameWidgets, PtRealizeWidget

PhabLookupWidget

`PhabLookupWidget` — searches a widget hierarchy for a named widget.

Syntax

`PhabLookupWidget (`*`hierarchy`*`, `*`name`*`, `*`prefix`*`)`

Arguments

`hierarchy`

A widget hierarchy, such as created by PhabCreateWidgets.

`name`

The name of an instance of the widget you are looking up, expressed as a symbol.

`prefix`

The widget name prefix, such as passed in a call to PhabCreateWidgets, or `nil` for none.

Returns

The requested widget, or `nil` on failure.

Description

This function searches a widget hierarchy for a widget, based on the *`name`* of an instance. The *`name`* must be entered as a symbol, not as a string. If any *`prefix`* is passed, it will be prepended to the *`name`*.

To use this function, you must load the PhabTemplate.lsp library with a call to **`require_lisp("PhabTemplate.lsp")`**.

Example

This example, `ex_PhabLookupWidget.g`, is included in the product distribution.

The following code:

```
#!/usr/cogent/bin/phgamma

PtInit(nil);
require_lisp("PhabTemplate.lsp");

file = PhabReadWidgetFile(string(_os_, "-WidgetFiles/wgt/loadwindow.wgtw"));
win = PhabCreateWidgets(file,nil,nil);

lookup = PhabLookupWidget(win, #MainPane, nil);
pretty_princ("Looked up and found:\n", lookup,"\n");
```

Generates the following output:

```
Looked up and found:
{PtPane (_callbacks) (_exceptions) (anchor_flags . 0x40000f00)
 (anchor_offsets . {PhRect (lr . {PhPoint (x . 0) (y . 0)})
      (ul . {PhPoint (x . 0) (y . 0)})})
 (area . {PhArea (pos . {PhPoint (x . 59) (y . 45)})
   (size . {PhDim (h . 225) (w . 281)})}) (basic_flags . 0x0)
 (border_width . 1) (bot_border_color . 0x606060) (color . 0x0)
 (container_flags . 0x404010) (cursor_color . 0xffffe0) (cursor_type . 0)
```

```
(dim . {PhDim (h . 225) (w . 281)}) (eflags . 0xffff8001)
(fill_color . 0xffdebd) (fill_pattern . ) (flags . 0x520) (focus)
(help_topic) (highlight_roundness . 0) (margin_height . 0)
(margin_width . 0) (name . MainPane) (pane_flags . 0x3b58c689)
(pos . {PhPoint (x . 59) (y . 45)}) (resize_flags . 0x0) (rid . 0)
(top_border_color . 0xffffff) (trans_pattern . ) (user_data)}
```

See Also

PhabReadWidgetFile, PhabReadWidget

PhabNameWidgets

`PhabNameWidgets` — assigns global variables to widget names.

Syntax

`PhabNameWidgets(`*`hierarchy`*`)`

Arguments

`hierarchy`

The widget hierarchy to which global variables will be assigned.

Returns

Doesn't return any useful value.

Description

This function walks a widget hierarchy and creates global variables of the same name as, and referring to, each of the widgets in the hierarchy that has a name.

To use this function, you must load the PhabTemplate.lsp library with a call to **`require_lisp("PhabTemplate.lsp")`**.

Example

This example, `ex_PhabNameWidgets.g`, is included in the product distribution.

```
#!/usr/cogent/bin/phgamma

/*
This example loads and displays a window, as in the example for
PhabLoad().  But here we use each function separately, including
PhabNameWidgets().
*/

PtInit(nil);
require_lisp("PhabTemplate.lsp");

file = PhabReadWidgetFile(string(_os_, "-WidgetFiles/wgt/loadwindow.wgtw"));
win = PhabCreateWidgets(file,nil,nil);
PhabNameWidgets(win);
PtRealizeWidget(MainWindow);

PtMainLoop();
```

See Also

PhabLoad

PhabReadWidget

`PhabReadWidget` — reads widgets (for internal use).

Syntax

```
PhabReadWidget (file)
```

Arguments

`file`

A file pointer to a previously opened file.

Returns

A widget pointer, as a list, or `nil`.

Description

This function is mainly used internally, supporting other functions such as `PhabReadWidgets` and `PhabReadWidgetFile`.

Example

This example, `ex_PhabReadWidget.g`, is included in the product distribution.

The following code:

```
#!/usr/cogent/bin/phgamma

/* This example reads the file: QNX4-WidgetFiles/wgt/readtestfile.wgtw
 * in QNX 4, and QNX6-WidgetFiles/wgt/readtestfile.wgtw in QNX 6.
 */

PtInit(nil);

fo = open(string(_os_, "-WidgetFiles/wgt/readtestfile.wgtw"),"r");
princ("The open file: \n", fo,"\n\n");

defs = PhabReadWidget(fo);
princ("PhabReadWidget gives you: \n", defs, "\n\n");
close(fo);
```

Generates the following output:

```
The open file:
#<File:WgtwReadFile/wgt/MyTestFile.wgtw>

PhabReadWidget gives you:
(0 PhAB113 1 nil)
```

See Also

PhabReadWidgetFile, PhabReadWidgets

PhabReadWidgetFile

PhabReadWidgetFile — loads and accesses widget definitions.

Syntax

```
PhabReadWidgetFile (filename)
PhabGetTop (definition)
PhabGetRoot (tree)
PhabGetChildren (tree)
PhabGetLevel (root)
PhabGetType (root)
PhabGetName (root)
PhabGetResources (root)
```

Arguments

filename

The name of the file to read (usually a .wgtw file), optionally including a path, written as a string.

definition

A widget definition as returned by a call to PhabReadWidgetFile.

tree

A widget definition tree.

root

A widget definition root as returned by a call to PhabGetRoot.

Returns

PhabReadWidgetFile returns a list of widget definitions, usually consisting of just one member, the first-level tree of widget definitions; or nil.

PhabGetTop returns the first tree of the widget definition hierarchy (which is usually the window and all the widgets it contains), or nil. This function effectively removes the outer parentheses of the definition list returned by PhabReadWidgetFile.

PhabGetRoot returns the definition of a widget without its children, or nil.

PhabGetChildren returns a list of the children of a widget definition tree, or nil.

The four remaining functions return the level, type, name, and resources of a widget, or nil.

Description

These functions are all related. Their pupose is to read a .wgtw file and access various parts of the widget definition hierarchy. This allows you to easily duplicate and reuse widgets (with or without their children) for various applications, using PhabCreateWidgets.

To use these functions, you must load the PhabTemplate.lsp library with a call to **require_lisp("PhabTemplate.lsp")**. To find the .wgtw file, PhabReadWidgetFile first searches the current directory, and if unsuccessful, it searches the require path (see the _require_path_ discussion in require).

A widget definition hierarchy has a regular structure, which is a nested hierarchy of lists. This allows easy access to any widget, using car, cdr, nth-car and nth-cdr. In fact, as you can see in the example, the PhabGet- functions are simply wrappers for combinations of car and cdr.

Table 1. Elements of a Widget Hierarchy

Element	Syntax	Description
Definition	(tree)	A list with one member, which is the main tree of the entire widget hierarchy.
Tree	(root [tree]...)	A list containing one root and zero or more trees.
Root	(level type name resources)	The definition of a single widget, without its children, expressed as a list.
Children	([tree]...)	A list of zero or more trees, without a root.
Level	integer	The level of the widget in the tree hierarchy.
Type	symbol	The type of widget.
Name	symbol	The name of the widget instance.
Resources	[((number . value)...)]	An association (assoc) list in which the first element of each pair is the photon resource number, and the second element is the value assigned to that resource.

Example

This example, `ex_PhabReadWidgetFile.g`, is included in the product distribution.

```
#!/usr/cogent/bin/phgamma

/*
 * This example reads and prints a widget file locating each widget and
 * printing its level, type, and name on one line, followed by its resources.
 * The program reads a file name from the command line, using the argument
 * (cadr(argv)). Our test file is a window that contains one button and
 * two panes, each of which also contain buttons. If no name is entered as an
 * argument on the command line, the default is the test window:
 * QNX4-WidgetFiles/wgt/readtestfile.wgtw for QNX 4, and
 * QNX6-WidgetFiles/wgt/readtestfile.wgtw for QNX 6.
 */

PtInit(nil);

/*
 * Define some PhabGet_ functions. (These are scheduled to be added as
 * library functions in the near future).
 */
PhabGetTop = car;
PhabGetRoot = car;
PhabGetChildren = cdr;
PhabGetLevel = car;
PhabGetType = cadr;
PhabGetName = caddr;
function PhabGetResources (wgt)
{
  caddr(cdr(wgt));
}

require_lisp("PhotonWidgets");
require_lisp("PhabTemplate");
```

```
/*
 * Read and print the complete widget file definition.
 */
if (cadr(argv))
  defs = PhabReadWidgetFile(cadr(argv));
else
  defs = PhabReadWidgetFile(string(_os_, "-WidgetFiles/wgt/readtestfile.wgtw"));

pretty_princ("The widget definition is:\n", defs, "\n\n");

tree = PhabGetTop(defs);

/*
 * Make a function that deconstructs the widget file definition,
 * extracting each widget definition in turn, and identifying
 * each part of each definition.
 */
function print_tree(tree)
{
  local root = PhabGetRoot(tree);
  local level = PhabGetLevel(root);
  local type = PhabGetType(root);
  local name = PhabGetName(root);
  local resources = PhabGetResources(root);

  princ("The ", name, " widget is type ", type, " at level ",
        level, ". \nIts resources are: \n");
  pretty_princ(resources, "\n");

  local children = PhabGetChildren(tree);
 if (children)
 {
  pretty_princ("Its children are:\n", children, "\n\n");
 }
 else princ("It has no children.\n\n");

  with child in children do {
    print_tree(child);
  }
}

/*
 * Call the function, then create and realize the widgets as an illustration.
 */
print_tree(tree);

NewWindow = car(car(PhabCreateWidgets(defs, nil, nil)));
NewWindow.SetPos(250,50);

PtRealizeWidget(NewWindow);

PtMainLoop();
```

Output:

```
    The widget definition is:
    (((1 PtWindow MyTestFile
         ((1005 . {PhPoint (x . 400) (y . 350)}) (2002 . 12632256)
          (2006 . 6316128) (18015 . Main Window) (1008 0 . -1)))
      ((2 PtButton MainWindowButton
          ((3011 . Press Main)
           (1007 . {PhPoint (x . 41) (y . 59)})
           (1005 . {PhPoint (x . 67) (y . 19)}))))
      ((2 PtPane PaneOne
          ((1005 . {PhPoint (x . 235) (y . 170)}) (1006 1280 . -1) (1001 . 1)
           (1007 . {PhPoint (x . 161) (y . 1)}) (2002 . 16768701)))
       ((3 PtButton PaneOneButtonA
           ((3011 . Press A)
```

```
 (1007 . {PhPoint (x . 61) (y . 31)})
 (1005 . {PhPoint (x . 48) (y . 19)})))))
   ((3 PtButton PaneOneButtonB
       ((3011 . Press B)
 (1007 . {PhPoint (x . 62) (y . 79)})
 (1005 . {PhPoint (x . 48) (y . 19)}))))))
  ((2 PtPane PaneTwo
      ((1005 . {PhPoint (x . 235) (y . 170)}) (1006 1280 . -1) (1001 . 1)
       (1007 . {PhPoint (x . 161) (y . 176)}) (2002 . 16760286)))
   ((3 PtButton PaneTwoButtonC
       ((3011 . Press C)
 (1007 . {PhPoint (x . 69) (y . 42)})
 (1005 . {PhPoint (x . 48) (y . 19)})))))))

The MyTestFile widget is type PtWindow at level 1.
Its resources are:
((1005 . {PhPoint (x . 400) (y . 350)}) (2002 . 12632256) (2006 . 6316128)
 (18015 . Main Window) (1008 0 . -1))
Its children are:
(((2 PtButton MainWindowButton
     ((3011 . Press Main)
      (1007 . {PhPoint (x . 41) (y . 59)})
      (1005 . {PhPoint (x . 67) (y . 19)})))))
 ((2 PtPane PaneOne
     ((1005 . {PhPoint (x . 235) (y . 170)}) (1006 1280 . -1) (1001 . 1)
      (1007 . {PhPoint (x . 161) (y . 1)}) (2002 . 16768701)))
  ((3 PtButton PaneOneButtonA
      ((3011 . Press A)
       (1007 . {PhPoint (x . 61) (y . 31)})
       (1005 . {PhPoint (x . 48) (y . 19)})))))
  ((3 PtButton PaneOneButtonB
      ((3011 . Press B)
       (1007 . {PhPoint (x . 62) (y . 79)})
       (1005 . {PhPoint (x . 48) (y . 19)}))))))
 ((2 PtPane PaneTwo
     ((1005 . {PhPoint (x . 235) (y . 170)}) (1006 1280 . -1) (1001 . 1)
      (1007 . {PhPoint (x . 161) (y . 176)}) (2002 . 16760286)))
  ((3 PtButton PaneTwoButtonC
      ((3011 . Press C)
       (1007 . {PhPoint (x . 69) (y . 42)})
       (1005 . {PhPoint (x . 48) (y . 19)})))))))

The MainWindowButton widget is type PtButton at level 2.
Its resources are:
((3011 . Press Main)
 (1007 . {PhPoint (x . 41) (y . 59)})
 (1005 . {PhPoint (x . 67) (y . 19)}))
It has no children.

The PaneOne widget is type PtPane at level 2.
Its resources are:
((1005 . {PhPoint (x . 235) (y . 170)}) (1006 1280 . -1) (1001 . 1)
 (1007 . {PhPoint (x . 161) (y . 1)}) (2002 . 16768701))
Its children are:
(((3 PtButton PaneOneButtonA
     ((3011 . Press A)
      (1007 . {PhPoint (x . 61) (y . 31)})
      (1005 . {PhPoint (x . 48) (y . 19)})))))
 ((3 PtButton PaneOneButtonB
     ((3011 . Press B)
      (1007 . {PhPoint (x . 62) (y . 79)})
      (1005 . {PhPoint (x . 48) (y . 19)}))))))

The PaneOneButtonA widget is type PtButton at level 3.
Its resources are:
((3011 . Press A)
 (1007 . {PhPoint (x . 61) (y . 31)})
 (1005 . {PhPoint (x . 48) (y . 19)}))
```

```
It has no children.

The PaneOneButtonB widget is type PtButton at level 3.
Its resources are:
((3011 . Press B)
 (1007 . {PhPoint (x . 62) (y . 79)})
 (1005 . {PhPoint (x . 48) (y . 19)}))
It has no children.

The PaneTwo widget is type PtPane at level 2.
Its resources are:
((1005 . {PhPoint (x . 235) (y . 170)}) (1006 1280 . -1) (1001 . 1)
 (1007 . {PhPoint (x . 161) (y . 176)}) (2002 . 16760286))
Its children are:
(((3 PtButton PaneTwoButtonC
     ((3011 . Press C)
      (1007 . {PhPoint (x . 69) (y . 42)})
      (1005 . {PhPoint (x . 48) (y . 19)}))))))

The PaneTwoButtonC widget is type PtButton at level 3.
Its resources are:
((3011 . Press C)
 (1007 . {PhPoint (x . 69) (y . 42)})
 (1005 . {PhPoint (x . 48) (y . 19)}))
It has no children.
```

See Also

PhabCreateWidgets

PhabReadWidgets

`PhabReadWidgets` — reads widget files (for internal use).

Syntax

```
PhabReadWidgets (file)
```

Arguments

`file`

A file pointer to a previously opened file.

Returns

A widget hierarchy, or `nil`.

Description

This function is used internally, supporting `PhabReadWidgetFile`. It takes the output from `PhabReadWidget` and combines it with another internal function that creates levels, making a widget hierarchy.

See Also

PhabReadWidgetFile, PhabReadWidget

PhabRoot

`PhabRoot` — a wrapper for the `car` function.

Syntax

`PhabRoot (`*`widget_hierarchy`*`)`

Arguments

`widget_hierarchy`

The name of an instantiated widget hierarchy, as returned from a call to `.PhabInstantiate`, which is a method of the class created by `PhabAttachWidgets`.

Returns

The root widget of the widget heirarchy.

Description

This is a convenience function that provides a more meaningful name for the `car` function, when used in the context of a widget hierarchy. Since Gamma handles widget hierarchies as lists, the car of the list is always the root widget.

See Also

`PhabChildren`

II. Graphics Functions

Table of Contents

ImageSubDivide

`ImageSubDivide` — divides an image into columns and rows.

Syntax

```
ImageSubDivide (image, cols, rows)
```

Arguments

`image`

An 8-bit image, instance of the PhImage class, such as returned by a call to PxLoadImage.

`cols`

The number of columns to divide the image into.

`rows`

The number of rows to divide the image into.

Returns

A list of images on success, otherwise `nil`.

Description

This function divides the passed *`image`* into the number of rows and columns specified, and puts the subimages into a list, by rows. That is, the first elements of the list are the subimages of row 1, then the subimages of row 2, etc.

To use this function, you must load the ImageProc.lsp library with a call to **`require_lisp("ImageProc.lsp")`**.

Example

This example, `ex_ImageSubDivide.g`, is included in the product distribution.

```
#!/usr/cogent/bin/phgamma

/*
This example puts up a window with a label image.  (Size: 96 by 96
pixels). Next to this image is the same image sub_divided into four
parts on four separate labels.
*/

PtInit(nil);
require_lisp("PhotonWidgets.lsp");
require_lisp("ImageProc.lsp");

win = new(PtWindow);
win.SetDim(300,160);

img = PxLoadImage("smiley.bmp");
lab = ImageToLabel(img);
lab.SetPos(25,25);

sub_img = (ImageSubDivide(img,2,2));

lab1 = ImageToLabel(car(sub_img));
lab1.SetPos(150,25);
```

```
lab2 = ImageToLabel(cadr(sub_img));
lab2.SetPos(210,25);

lab3 = ImageToLabel(caddr(sub_img));
lab3.SetPos(150,85);

lab4 =ImageToLabel(caddr(cdr(sub_img)));
lab4.SetPos(210,85);

PtRealizeWidget(win);
PtMainLoop();
```

See Also

PxLoadImage

ImageThumbNail

`ImageThumbNail` — resizes images by pixels.

Syntax

`ImageThumbNail (`*`image`*`, `*`wgoal`*`, `*`hgoal`*`)`

Arguments

`image`

An 8-bit image, instance of the PhImage class, such as returned by a call to PxLoadImage.

`wgoal`

The desired width of the thumbnail, in pixels.

`hgoal`

The desired height of the thumbnail, in pixels.

Returns

An image on success, otherwise `nil`.

Description

This function reduces or enlarges an image to the height and width specified. Although both height and width must be specified, `ImageThumbNail` always reduces or enlarges proportionally, using the smaller of two size parameters.

To use this function, you must load the ImageProc.lsp library with a call to **`require_lisp("ImageProc.lsp")`**.

Example

This example, `ex_ImageThumbNail.g`, is included in the product distribution.

```
#!/usr/cogent/bin/phgamma

/*
This example puts up a window with a label image.  (Size: 96 by 96
pixels). Next to this image is the same image reduced by 1/4 and 3/4.
In the second reduction, arguments were passed for a dispropotional
result, but ImageThumbNail() chooses the smaller of the two.
*/

PtInit(nil);
require_lisp("PhotonWidgets.lsp");
require_lisp("ImageProc.lsp");

win = new(PtWindow);
win.SetDim(250,200);

img = PxLoadImage("smiley.bmp");
lab = ImageToLabel(img);
lab.SetPos(25,25);

imgt1 = (ImageThumbNail(img,24,24));
labelt1 = ImageToLabel(imgt1);
labelt1.SetPos(150,25);
```

```
imgt2 = (ImageThumbNail(img,72,200));
labelt2 = ImageToLabel(imgt2);
labelt2.SetPos(150,80);

PtRealizeWidget(win);
PtMainLoop();
```

See Also

PxReduceImage

ImageToLabel

`ImageToLabel` — makes a label from an image.

Syntax

`ImageToLabel (`*`image`*`)`

Arguments

`image`

The image of which to make a label.

Returns

An image label on success, otherwise `nil`.

Description

This function creates a label, strips off its borders, converts it to an image label, and applies the passed *`image`* to it.

To use this function, you must load the ImageProc.lsp library with a call to **`require_lisp("ImageProc.lsp")`**.

Example

This example, `ex_ImageToLabel.g`, is included in the product distribution.

```
#!/usr/cogent/bin/phgamma

//This example puts up a window with an image label in it.

PtInit(nil);
require_lisp("PhotonWidgets.lsp");
require_lisp("ImageProc.lsp");

win = new(PtWindow);
win.SetDim(140,140);

img = PxLoadImage("smiley.bmp");
lab = ImageToLabel(img);
lab.SetPos(25,25);

PtRealizeWidget(win);
PtMainLoop();
```

PgCMY

PgCMY — represents colors as combinations of cyan, magenta, and yellow.

Syntax

PgCMY(*cyan_component, magenta_component, yellow_component*)

Arguments

cyan_component

The cyan component of the desired color expressed as a number between 0-255 decimal (0x00-0xff hex).

magenta_component

The magenta component of the desired color expressed as a number between 0-255 decimal (0x00-0xff hex).

yellow_component

The yellow component of the desired color expressed as a number between 0-255 decimal (0x00-0xff hex).

Returns

A 24-bit number in the range 0x000000 - 0xffffff that represents the combination of the *cyan_component*, the *magenta_component*, and the *yellow_component* using the CMY color model.

Description

The CMY color model (classically known as the CMYK model) is a popular representation, especially for pigment-based, subtractive color processes like printing. The RGB color model, on the other hand, is additive. The RGB model is light-based, and works on adding color to black. For example, adding green light to blue light gives yellow light. Adding red, green, and blue light together gives white light. So, in the RGB model, the full value is white, represented as (255,255,255).

The CMY model, though, is subtractive, since it is pigment-based. Pigments absorb all light except the color reflected. Thus when you add pigments, you actually subtract color. For example, adding cyan, magenta, and yellow pigments will yield black. That means the full value of the CMY color model is black, with a value of (255,255,255).

Example

```
Gamma> WHITE = PgCMY(0,0,0);
0xffffff
Gamma> YELLOW = PgCMY(0,0,255);
0xffff00
Gamma> ORANGE = PgCMY(0,90,255);
0xffa500
Gamma> RED = PgCMY(0,255,255);
0xff0000
Gamma> MAGENTA = PgCMY(0,255,0);
0xff00ff
Gamma> BLUE = PgCMY(255,255,0);
0xff
Gamma> GREEN = PgCMY(255,0,255);
0xff00
```

```
Gamma> BLACK = PgCMY(255,255,255);
0x0
```

See Also

PgRGB, PgGray

and in Photon documentation: PgCMY.

PgGray

PgGray — converts a gray level index from 8-bit to 24-bit color.

Syntax

```
PgGray (gray_level)
```

Arguments

gray_level

A level for gray between 0 - 255 decimal (0x00-0xff hex).

Returns

A 24-bit color representation of the *gray_level*.

Description

This function converts a gray level index from 8-bit to 24-bit color. For example, an image may contain 256 shades of gray. To convert those grays to 24-bit color use this function to acquire the 24-bit equivalents.

Example

```
Gamma> BLACK = PgGray(0);
0x0
Gamma> WHITE = PgGray(255);
0xffffff
Gamma> MEDGRAY = PgGray(128);
0x808080
Gamma> WHITE = PgRGB(255,255,255);
0xffffff
Gamma> PgGray(0xff) == WHITE;
t
```

See Also

PgGrayValue

and in Photon documentation: PgGray.

PgGrayValue

`PgGrayValue` — converts a color to a gray-scale equivalent.

Syntax

```
PgGrayValue (color)
```

Arguments

`color`

A 24-bit color in hexadecimal format in the range 0x000000 - 0xffffff.

Returns

An 8-bit (0x00 - 0xff hex) gray-scale representation of the *`color`*.

Description

This function converts a 24-bit color to an 8-bit gray-scale equivalent.

Example

```
Gamma> PgGrayValue (0xffffff);
0xff
Gamma> PgGrayValue (0x808080);
0x80
Gamma> PgGrayValue (PgRGB(255,0,0));
0x4c
```

See Also

PgGray

and in Photon documentation: PgGrayValue.

PgHSV

PgHSV — represents color by hue, saturation, and brightness.

Syntax

```
PgHSV (hue, saturation, value)
```

Arguments

hue

A numeric representation of the hue component (0-255 decimal, 0x00-0xff hex).

saturation

A numeric representation of the saturation component (0-255 decimal, 0x00-0xff hex).

value

A numeric representation of the value (often called brightness) component (0-255 decimal, 0x00-0xff hex).

Returns

A 24-bit color in the range 0x000000 - 0xffffff that is the numeric representation of the *hue*, *saturation*, and *value* (brightness).

Description

The hue, saturation, and brightness color system is another color model that is popular. The hue is the color, such as yellow, orange, purple, green, etc. The saturation is a range from white to the fully saturated color. The value (or brightness) is a range from black to the fully saturated color.

Example

```
Gamma> PgHSV(0,0,0);
0x00
Gamma> PgHSV(0,0,255);
0xffffff
Gamma> PgHSV(0,0,128) == PgRGB(128,128,128);
t
```

See Also

PgRGB, PgCMY

and in Photon documentation: PgHSV.

PgRedValue, PgGreenValue, PgBlueValue

`PgRedValue, PgGreenValue, PgBlueValue` — indicate color strength.

Syntax

```
PgRedValue(color)
PgGreenValue(color)
PgBlueValue(color)
```

Arguments

`color`

A 24-bit color number usually held as a hexadecimal number between 0x000000 (black) and 0xffffff (white).

Returns

A numeric representation of the red, green, or blue portion of the *`color`*.

Description

The return value is a number between 0x00 and 0xff (0-255 decimal) indicating the strength of the blue, red, or green portion of the *`color`*. In Photon, colors are held as 24-bit numbers. The most common representation of these numbers is a hex number of the form: `0xrrggbb` where `rr` is a hex value of red, `gg` is the hex value of green, and `bb` is the hex value of blue.

Example

```
Gamma> color = PgRGB(77,0,128);
0x4d0080
Gamma> PgBlueValue(color);
0x80
Gamma> PgRedValue(color);
0x4d
Gamma> PgGreenValue(color);
0x0
```

See Also

PgRGB, PgCMY, PgGrayValue, PgHSV

and in Photon documentation: PgRedValue, PgGreenValue, PgBlueValue.

PgRGB

PgRGB — represents colors as combinations of red, green, and blue.

Syntax

```
PgRGB (red, green, blue)
```

Arguments

red

A number 0-255 decimal (0x00-0xff hex) representing the red component of the color.

green

A number 0-255 decimal (0x00-0xff hex) representing the green component of the color.

blue

A number 0-255 decimal (0x00-0xff hex) representing the blue component of the color.

Returns

A 24-bit color in the range 0x0 - 0xffffff that represents the combination of the passed args using the RGB color model.

Description

The RGB color model is the most familiar since it uses additive color combinations. As it is based on light, the lowest value is (0,0,0), or black (no light, no color); and the highest value is (255,255,255), or white (a combination of all colors).

Example

```
Gamma> WHITE = PgRGB(255,255,255);
0xffffff
Gamma> YELLOW = PgRGB(0,255,255);
0xffff
Gamma> ORANGE = PgRGB(255,165,0);
0xffa500
Gamma> RED = PgRGB(255,0,0);
0xff0000
Gamma> MAGENTA = PgRGB(255,0,255);
0xff00ff
Gamma> BLUE = PgRGB(0,0,255);
0xff
Gamma> GREEN = PgRGB(0,255,0);
0xff00
Gamma> BLACK = PgRGB(0,0,0);
0x0
```

See Also

PgCMY, PgHSV, PgRedValue, PgBlueValue, PgGreenValue

and in Photon documentation: PgRGB.

PxConvolveImage

PxConvolveImage — performs matrix-based convolutions on images.

Syntax

```
PxConvolveImage (image, matrix)
```

Arguments

image

A PhImage, previously grey-scaled with a call to PxGreyImage.

matrix

Usually a 3 X 3 matrix, entered as an array of arrays in the format:

```
[[v1 v2 v3]
 [v4 v5 v6]
 [v7 v8 v9]]
```

Returns

A PhImage.

Description

This function is used to convolve images. It applies the *matrix* to every point of the image, allowing for such effects as edge enhancement and image sharpening.

Example

This example, ex_PxConvolveImage.g, is included in the product distribution.

```
#!/usr/cogent/bin/phgamma

/*
The following example demonstrates PxCopyImage(), PxGreyImage(),
and PxConvolveImage().  An image is loaded and converted to grey
scales. Then it is convolved for image sharpening, vertical and
horizontal edge enhancement, and laplace edge detection.
*/

PtInit(nil);
require_lisp("PhotonWidgets.lsp");

win = new(PtWindow);

// Load and display the image.
pic1 = PxLoadImage("taj.bmp");
lab1 = new(PtLabel);
lab1.label_type = Pt_IMAGE;
lab1.label_data = pic1;

// Copy the image and convert the colors to grey scales.
pic2 = PxCopyImage(pic1);
pic2 = PxGreyImage(pic2);
lab2 = new(PtLabel);
lab2.label_type = Pt_IMAGE;
lab2.label_data = pic2;
```

```
lab2.SetPos(300,0);

// Copy the grey_scaled image and convolve for image sharpening.
matrix = array(array(0,-1,0),
        array(-1,5,-1),
        array(0,-1,0));
pic3 = PxCopyImage(pic2);
pic3 = PxConvolveImage(pic3, matrix);
lab3 = new(PtLabel);
lab3.label_type = Pt_IMAGE;
lab3.label_data = pic3;
lab3.SetPos(600,0);

// Copy the grey_scaled image and convolve for vertical edge enhancement.
matrix = array(array(2,0,-2),
        array(2,0,-2),
        array(2,0,-2));
pic4 = PxCopyImage(pic2);
pic4 = PxConvolveImage(pic4, matrix);
lab4 = new(PtLabel);
lab4.label_type = Pt_IMAGE;
lab4.label_data = pic4;
lab4.SetPos(0,200);

// Copy the grey_scaled image and convolve for horizontal edge enhancement.
matrix = array(array(2,2,2),
        array(0,0,0),
        array(-2,-2,-2));
pic5 = PxCopyImage(pic2);
pic5 = PxConvolveImage(pic5, matrix);
lab5 = new(PtLabel);
lab5.label_type = Pt_IMAGE;
lab5.label_data = pic5;
lab5.SetPos(300,200);

// Copy the grey_scaled image and convolve for laplace edge detector.
matrix = array(array(1,1,1),
        array(1,-8,1),
        array(1,1,1));
pic6 = PxCopyImage(pic2);
pic6 = PxConvolveImage(pic6, matrix);
lab6 = new(PtLabel);
lab6.label_type = Pt_IMAGE;
lab6.label_data = pic6;
lab6.SetPos(600,200);

PtRealizeWidget(win);
PtMainLoop();
```

See Also

PxCopyImage, PxGreyImage, PxThresholdImage

PxCopyImage

PxCopyImage — copies an image.

Syntax

```
PxCopyImage (image)
```

Arguments

image

PhImage.

Returns

A copy of the *image*.

Example

See the example for PxConvolveImage

See Also

PxGreyImage, PxConvolveImage, PxThresholdImage,

PxGreyImage

PxGreyImage — converts colored images to grey.

Syntax

```
PxGreyImage (image)
```

Arguments

image

PhImage.

Returns

The *image*, grey-scaled.

Description

This convenience function converts the colors of an image to grey scales, by calling PgGray and PgGrayValue.

Example

See the example for PxConvolveImage

See Also

PxCopyImage, PxConvolveImage, PxThresholdImage,

PxLoadImage

`PxLoadImage` — loads image files.

Syntax

```
PxLoadImage (filename)
```

Arguments

`filename`

A string containing a path to a valid image file.

Returns

The image, as an instance of PhImage.

Description

This function loads .BMP, .JPG, and .GIF files and returns the image file.

Example

This example, `ex_PxLoadImage.g`, is included in the product distribution.

```
#!/usr/cogent/bin/phgamma

/*
The following program creates a new window with a label.  The label
type is changed from text (default) to image and a pre_loaded picture
is put into the label.
*/

PtInit(nil);

win = new(PtWindow);

pic = PxLoadImage("smiley.bmp");

lab = new(PtLabel);
lab.label_type = Pt_IMAGE;
lab.label_data = pic;

PtRealizeWidget(win);
PtMainLoop();
```

See Also

PxReduceImage

and in Photon documentation: PxLoadImage.

PxReduceImage

PxReduceImage — resizes images by percentage.

Syntax

```
PxReduceImage (image, xscale, yscale)
```

Arguments

image

An image.

xscale

The x-scale resize factor.

yscale

The y-scale resize factor.

Returns

The resized image.

Description

This function is used to resize images. The *xscale* and *yscale* are percentages of 1. Entries less than 1 reduce the image, those greater than 1 enlarge it. Images can be resized in differing proportions on the x and y axes, which permits image distortion.

Example

This example, ex_PxReduceImage.g, is included in the product distribution.

```
#!/usr/cogent/bin/phgamma

/*
This example shows three images in a window.  The first image is full
size, the second is reduced on the x and y axes, and the third is
enlarged on the x axis and reduced on the y axis.
*/

PtInit(nil);
require_lisp("PhotonWidgets");

win = new(PtWindow);

pic = PxLoadImage("smiley.bmp");

lab = new(PtLabel);
lab.label_type = Pt_IMAGE;
lab.label_data = pic;

lab2 = new(PtLabel);
lab2.label_type = Pt_IMAGE;
lab2.label_data = PxReduceImage(pic,.5,.5);
lab2.SetPos(100,0);

lab3 = new(PtLabel);
lab3.label_type = Pt_IMAGE;
```

```
lab3.label_data = PxReduceImage(pic,2,.25);
lab3.SetPos(100,60);

PtRealizeWidget(win);
PtMainLoop();
```

See Also

PxLoadImage, ImageThumbNail

PxSubImage

PxSubImage — returns a sub-image of a passed image.

Syntax

PxSubImage (*image*, *x*, *y*, *width*, *height*)

Arguments

image

An image as a buffer.

x

The starting x coordinate.

y

The starting y coordinate.

width

The width (in pixels) of the sub-image to take.

height

The height (in pixels) of the sub-image to take.

Returns

A buffer containing the sub-image.

Description

This function returns a sub-image of the *image* starting at pixel coordinate (*x*,*y*) and of the passed *width* and *height*.

Example

This example, ex_PxSubImage.g, is included in the product distribution.

```
#!/usr/cogent/bin/phgamma

require_lisp("PhotonWidgets.lsp");
require_lisp("PhabTemplate.lsp");
PtInit(nil);

win = new(PtWindow);

pic = PxLoadImage("smiley.bmp");

lab = new(PtLabel);
lab.label_type = Pt_IMAGE;
lab.label_data = pic;

lab2 = new(PtLabel);
lab2.SetArea(120,0,100,100);
lab2.label_type = Pt_IMAGE;
lab2.label_data = PxSubImage(pic,50,20,30,60);

PtRealizeWidget(win);
PtMainLoop();
```

See Also

PxLoadImage, PxReduceImage

PxThresholdImage

PxThresholdImage — converts grey-scale images to black and white.

Syntax

```
PxThresholdImage (image, threshold, low?, high?)
```

Arguments

image

PhImage, previously grey-scaled with a call to PxGreyImage.

threshold

An integer that determines the threshold level between black and white.

low

Optional integer setting for black values, default = 0.

high

Optional integer setting for white values, default = -1.

Returns

A PhImage.

Description

This function converts greys to black and white, at a given threshold point. As the *threshold* increases, the image darkens.

Example

This example, ex_PxThresholdImage.g, is included in the product distribution.

```
#!/usr/cogent/bin/phgamma

/*
The following example demonstrates PxCopyImage(), PxGreyImage(),
and PxThresholdImage().  An image is loaded and converted to grey
scales.  Then four threshold images are created.
*/

PtInit(nil);
require_lisp("PhotonWidgets.lsp");

win = new(PtWindow);

// Load and display the image.
pic1 = PxLoadImage("taj.bmp");
lab1 = new(PtLabel);
lab1.label_type = Pt_IMAGE;
lab1.label_data = pic1;

// Copy the image and convert the colors to grey scales.
pic2 = PxCopyImage(pic1);
pic2 = PxGreyImage(pic2);
lab2 = new(PtLabel);
lab2.label_type = Pt_IMAGE;
lab2.label_data = pic2;
```

```
lab2.SetPos(300,0);

// Copy the grey_scale and make a threshold image
// with a medim_level threshold and reversed high/low.
pic3 = PxCopyImage(pic2);
pic3 = PxThresholdImage(pic3, 150, -2, 1);
lab3 = new(PtLabel);
lab3.label_type = Pt_IMAGE;
lab3.label_data = pic3;
lab3.SetPos(600,0);

// Copy the grey_scale and make a threshold image
// with a low_level threshold.
pic4 = PxCopyImage(pic2);
pic4 = PxThresholdImage(pic4, 100, 1, -2);
lab4 = new(PtLabel);
lab4.label_type = Pt_IMAGE;
lab4.label_data = pic4;
lab4.SetPos(0,200);

// Copy the grey_scale and make a threshold image
// with a medium_level threshold.
pic5 = PxCopyImage(pic2);
pic5 = PxThresholdImage(pic5, 150, 1, -2);
lab5 = new(PtLabel);
lab5.label_type = Pt_IMAGE;
lab5.label_data = pic5;
lab5.SetPos(300,200);

// Copy the grey_scale and make a threshold image
// with a high_level threshold.
pic6 = PxCopyImage(pic2);
pic6 = PxThresholdImage(pic6, 200, 1, -2);
lab6 = new(PtLabel);
lab6.label_type = Pt_IMAGE;
lab6.label_data = pic6;
lab6.SetPos(600,200);

PtRealizeWidget(win);
PtMainLoop();
```

See Also

PxCopyImage, PxGreyImage, PxConvolveImage

III. Photon Functions

Table of Contents

PhGetRects

PhGetRects — gets rectangle sets.

Syntax

```
PhGetRects (event)
```

Arguments

event

An instance of the PhEvent class.

Returns

The rectangle associated with the *event*.

Description

This function returns two corner points that define a rectangle set associated with the *event*. The output is a list containing a single instance of the PhRect class. The PhRect class has two members, the upper left corner (ul) and lower right corner (lr). Each of these members contains a dotted list of the name (ul or lr) and instances of the PhPoint class.

Example

This example, ex_PhGetRects.g, is included in the product distribution.

Running the following code:

```
#!/usr/cogent/bin/phgamma

/*
This example prints the location of the mouse pointer when the mouse
button is clicked or the mouse moves. The rectangle is a single point,
so the upper left corner of the rectangle is equal to the lower right
corner.
*/

PtInit(nil);

win = new(PtWindow);
PtRealizeWidget(win);

function cb_function ()
{
    princ(PhGetRects(cbinfo.event),"\n");
}

PtAttachCallback(win,Pt_CB_RAW,#cb_function());

PtMainLoop();
```

produces the following kind of output.

```
    ({PhRect (lr . {PhPoint (x . 64) (y . 42)}) (ul . {PhPoint (x . 64) (y . 42)})})
```

See Also

In Photon documentation: PhRect.

PhEmitEvent

`PhEmitEvent` — emits an event.

Syntax

```
PhEmitEvent (event, rectangle, event_data)
```

Arguments

`event`

A PhEvent.

`rectangle`

A PhRect.

`event_data`

Variable-length, event-specific data.

Returns

0 on success, or -1 indicating no pending events or error.

Description

This function emits an event.

See Also

In Photon documentation: PhEventEmit.

PhInitDrag

`PhInitDrag` — starts a drag.

Syntax

`PhInitDrag (rid, flags, rect, boundary?, input_group?, min?, max?, step?)`

Arguments

`rid`

The region id of *`rect`* and *`boundary`*.

`flags`

Options: track_left, track_right, track_top, track_bottom, track_drag, drag_track, key_motion, drag_nobutton control drag behavior.

`rect`

A PhRect area to be dragged.

`boundary`

A PhRect boundary of the dragging operation.

`input_group`

The input group number (optional).

`min`

The PhDim of the minimum rectangle size (optional).

`max`

The PhDim of the maximum rectangle size (optional).

`step`

A PhDim for the granularity of the drag (optional).

Returns

0 on success, or -1 on error.

Description

This function starts a drag of the object defined by *`rect`*, within the *`boundary`*. Permissible *`flags`* include: drag_track, track_left, track_right, track_top, track_bottom, track_drag.

See Also

In Photon documentation: PhInitDrag, Dragging in the Events chapter of the Progammer's Guide.

PhMoveCursor

`PhMoveCursorAbs, PhMoveCursorRel` — move the cursor to an absolute or relative position.

Syntax

```
PhMoveCursorAbs (input_group, x, y)
PhMoveCursorRel (input_group, x, y)
```

Arguments

input_group

The input group number.

x

The x-coordinate.

y

The y-coordinate.

Returns

`t`.

Description

PhMoveCursorAbs moves the cursor for the *input_group* to the absolute coordinates *x* and *y*.

PhMoveCursorRel moves the cursor for the *input_group* a distance of *x* and *y* from its current position.

See Also

In Photon documentation: PhMoveCursorAbs, PhMoveCursorRel.

PhQueryRids

`PhQueryRids` — finds selected regions.

Syntax

```
PhQueryRids (flags, rid, input_group, type, sense, emitter, rect, maxrids)
```

Arguments

`flags`

Options: ridquery_ig_pointer, ridquery_toward.

`rid`

A region id number.

`input_group`

An input group number.

`type`

A region type.

`sense`

An event type.

`emitter`

The region from which the query originates.

`rect`

A PtRect.

`maxrids`

The maximum number of region ids to return.

Returns

The number of regions found, or -1 on error.

Description

This function finds all the regions that meet the specified criteria. Only those regions that satisfy criteria of the *`flag`*, *`rid`*, *`input_group`*, *`type`*, and *`sense`*, and also intersect the area of *`rect`*, are included.

See Also

In Photon documentation: PhQueryRids.

PhRegionChange

`PhRegionChange` — changes the definition of a region.

Syntax

`PhRegionChange (`*`fields`*`, `*`flags`*`, `*`info`*`, `*`rect`*`)`

Arguments

`fields`

The fields in *`info`* that are to be changed.

`flags`

Options: expose_region OR expose_family.

`info`

The information structure of the region.

`rect`

The PhRect associated with the region.

Returns

0 on success, else -1 on error.

Description

This function changes the definition of the region. The *`flags`* control what to do if part of the region gets exposed, sending an ev_expose event to the region, to the region's descendents, or both.

See Also

In Photon documentation: PhRegionChange.

PhRegionQuery

`PhRegionQuery` — gets information about a region.

Syntax

```
PhRegionQuery (rid)
```

Arguments

`rid`

The region id.

Returns

A list in the format (PhRegion PhRect).

Description

This function gets information about a region, using its region id number. It returns the region as a `PhRegion` widget and its associated area as a `PhRect` widget, as a list.

See Also

In Photon documentation: PhRegionQuery.

PhWindowQueryVisible

`PhWindowQueryVisible` — finds the visible extent of a window.

Syntax

```
PhWindowQueryVisible (flags, rid, input_group)
```

Arguments

`flags`

One of: query_graphics, query_input_group, query_exact, query_ig_pointer, query_ig_region.

`rid`

The region id.

`input_group`

The input group number.

Returns

The area as a PhRect.

Description

This function calculates the visible extent, based on the region type passed in *`flag`*, for every region intersecting with *`rid`*. One and only one *`flag`* must be set, but the *`rid`* can be 0.

The *`input_group`* should be the input_group field for an event, if there is any. Otherwise, use the value of the PHIG environment, if defined. If neither of these is available, use 1.

See Also

In Photon documentation: PhWindowQueryVisible.

IV. Widget Functions

Table of Contents

PtAddResource

PtAddResource — adds a resource.

Syntax

```
PtAddResource (resnum, name)
```

Arguments

resnum

An integer resource number.

name

A character string resource name.

Returns

The *resnum*.

Description

This function adds new resources. The first time a resource is added, an array is created internally to catalog it. Each additional resource is cataloged in the array. If the passed *resnum* already belongs to an existing resource, the same *resnum* is returned, but PtAddResource leaves the existing resource intact and does not add anything.

PtAttachCallback

PtAttachCallback — associates an action with a Photon event.

Syntax

```
PtAttachCallback (widget, callback_number, callback_function,
eventmask?|key_cap?, key_mods?, key_flags?)
```

Arguments

widget

The widget to attach the callback function on.

callback_number

The callback to trigger on.

callback_function

The function to call when the event occurs.

eventmask

An optional filter, for the callback only.

callbacks

Those that match this eventmask trigger the function.

keycap

Optional keyboard capture ID.

key_mods

Optional keyboard modifiers to the keycap (e.g. SHIFT, CTRL, ALT).

key_flags

Optional key flags (e.g. key up, key down, key repeat).

Returns

t when successful.

Description

The PtAttachCallback function is one of the most useful functions to use with Photon in Gamma. This functions allows an action to be associated with a Photon event.

The *callback_number* is usually expressed as a constant. For example, Pt_CB_ACTIVATE, Pt_CB_RAW, Pt_CB_SELECTION. A list of the available callbacks is given with each widget definition in the widget reference.

The *callback_function* is protected using quote operators (#, ‘, and @) and is evaluated only when the callback occurs.

The *eventmask* is used to look for specific events. For example, the callback number Pt_CB_BUT_PRESS can be masked with the event mask constant Pt_EV_BUT_RELEASE so that the callback function is only called when the mouse button is released.

Example

This example, `ex_PtAttachCallback.g`, is included in the product distribution.

```
#!/usr/cogent/bin/phgamma

require_lisp("PhotonWidgets.lsp");
PtInit(nil);

win = new(PtWindow);
win.SetDim(200,50);

function pressed (held)
{
  if (!held)
    widget.text_string = string("Pressed: cb called ",counter++," times");
  else
    widget.text_string = string("Held: cb called ",counter++," times");
}

counter = 1;
button = new(PtButton);
button.SetPos(10, 10);
button.text_string = "------ Press Me ------";
PtAttachCallback(button,Pt_CB_ACTIVATE,#pressed(nil));
PtAttachCallback(button,Pt_CB_REPEAT,#pressed(t));

PtRealizeWidget(win);
PtMainLoop();
```

See the Callback Functions section of the Photon Functions chapter (Tutorial One), or the Photon: the Controller window of the Controller Functions chapter (Tutorial Two) in the Cogent Tools Demo and Tutorials book for an example of this function used in context.

See Also

PtCallbackInfo, PtRemoveCallback

PtCollideWidget

`PtCollideWidget` — checks for collisions between widgets.

Syntax

`PtCollideWidget(`*`widget`*`, `*`target_list`*`, `*`flags`*`?)`

Arguments

`widget`

The widget to check for a collision.

`target_list`

A list of widgets that the first argument could possibly collide with.

`flags`

nil

Returns

A list containing the definitions of the colliding widgets, if any. Otherwise, `nil`.

Description

This function checks one widget against a list of widgets to see if it occupies the same area as any of them.

Example

```
Gamma> require_lisp("PhotonWidgets");
"/usr/cogent/lib/PhotonWidgets.lsp"
Gamma> PtInit(nil);
t
Gamma> win = new(PtWindow);
window instance definition
Gamma> ellipse1 = new(PtEllipse);
ellipse instance def
Gamma> ellipse1.SetArea(0,0,100,100);
position instance
Gamma> ellipse2 = new(PtEllipse);
ellipse instance def
Gamma> ellipse2.SetArea(100,100,100,100);
position structure instance
Gamma> PtRealizeWidget(win);
t
Gamma> PtCollideWidget(ellipse1, list(ellipse2));
nil
Gamma> ellipse1.SetPos(50,50);
position structure instance
Gamma> PtCollideWidget(ellipse1, list(ellipse2));
list containing definition for ellipse2
```

PtContainerGiveFocus

PtContainerGiveFocus — gives focus to a widget.

Syntax

PtContainerGiveFocus (*widget*, *event*)

Arguments

widget

A container-type widget.

event

The event to pass to both the object losing focus and the widget gaining focus.

Returns

t if successful.

Description

This function gives focus to the *widget*. The *event* argument is used to pass a known event to both the object losing focus and the *widget*, which is gaining focus. If the event is not important to you, pass a new instance of the PhEvent class.

Example

This example, ex_PtContainerGiveFocus.g, is included in the product distribution.

```
#!/usr/cogent/bin/phgamma

/*
This example puts up a window with 10 button widgets.  The focus is
moved through the buttons by a timer calling the change_focus()
function every 0.2 sec.  The appearance of the buttons (a rendered
focus) changes when they receive focus.
*/

require_lisp("PhotonWidgets.lsp");
PtInit(nil);
win = new(PtWindow);

max = 10;
button_array = make_array(0);
for(i=0;i<max;i++)
{
  button_array[i] = new(PtButton);
  button_array[i].text_string = string("But",i);
  button_array[i].SetPos((i % 5) * 35, div(i,5) * 30);
  PtRealizeWidget(button_array[i]);
}
PtRealizeWidget(win);

current_focus = 0;
function change_focus ()
{
  PtContainerGiveFocus(button_array[current_focus],new(PhEvent));
  current_focus = (current_focus + 1) % max;
}
```

```
every(0.2,#change_focus());
PtMainLoop();
```

See Also

In Photon documentation: PtContainerGiveFocus.

PtContainerHold, PtContainerRelease

PtContainerHold, PtContainerRelease — prevent/allow updating of container-widget contents.

Syntax

```
PtContainerHold (widget)
PtContainerRelease (widget)
```

Arguments

widget

A PtContainer widget.

Returns

t if successful.

Description

PtContainerHold prevents the updating of contents of the *widget*. PtContainerRelease allows updating to continue.

Successive calls to PtContainerHold increment a counter. An equal number of calls to PtContainerRelease are required to decrement the counter so the *widget* can be updated.

Example

This example, ex_PtContainerHold.g, is included in the product distribution.

```
#!/usr/cogent/bin/phgamma

/*  This sample program puts two panes in a window.
    Each pane contains a label that is updated every
    second by a timer.  A button toggles the Held and
    Released state for each pane.
*/

require_lisp("PhotonWidgets");
PtInit(nil);
win=new(PtWindow);

class_add_ivar(PtPane,#held,nil);

but1 = new(PtButton);
but1.text_string = "Released";
but1.SetPos(20,40);

but2 = new(PtButton);
but2.text_string = "Released";
but2.SetPos(20,140);

pane1 = new(PtPane);
pane1.fill_color = PgRGB(160,160,160);
pane1.SetArea(100,0,200,100);

time1 = new(PtLabel);
time1.text_string = date();
```

```
PtSetParentWidget(win);
pane2 = new(PtPane);
pane2.fill_color = PgRGB(220,220,220);
pane2.SetArea(100,100,200,100);

time2 = new(PtLabel);
time2.text_string = date();

every(1,#time1.text_string = date());
every(1,#time2.text_string = date());

function toggle_hold (pane)
{
  if (pane.held)
  {
    PtContainerRelease(pane);
    pane.held = nil;
    widget.text_string = "Released";
  }
  else
  {
    PtContainerHold(pane);
    pane.held = t;
    widget.text_string = "Held";
  }
}

PtAttachCallback(but1,Pt_CB_RAW,#toggle_hold(pane1),Ph_EV_BUT_PRESS);
PtAttachCallback(but2,Pt_CB_RAW,#toggle_hold(pane2),Ph_EV_BUT_PRESS);

PtRealizeWidget(win);

PtMainLoop();
```

See Also

In Photon documentation: PtContainerRelease, PtContainerHold.

PtDestroyWidget

`PtDestroyWidget` — deletes a widget.

Syntax

```
PtDestroyWidget (widget)
```

Arguments

`widget`

The widget to destroy.

Returns

`t` if successful, otherwise an error message.

Description

This function deletes the *`widget`*. It will remove all callbacks attached to it as well. If the widget class has a defined destructor, then it is called. The class instance object is marked as a destroyed instance type and the memory used for the *`widget`* is recovered the next time the garbage collector runs.

Example

```
Gamma> PtInit(nil);
t
Gamma> win = new(PtWindow);
window instance definition
Gamma> PtDestroyWidget(win);
t
```

See Also

PtRealizeWidget, PtUnrealizeWidget

and in Photon documentation: PtDestroyWidget.

PtEventHandler

`PtEventHandler` — is not yet implemented.

Syntax

`PtEventHandler (`*`event`*`)`

PtExtentWidget

PtExtentWidget — forces a widget to calculate its extent.

Syntax

```
PtExtentWidget (widget)
```

Arguments

widget

The widget that will calculate its extent.

Returns

t if successful.

Description

This function forces the *widget* to calculate its extent. When calculating its extent, a widget will resize itself to contain all of its children.

Example

This example, ex_PtExtentWidget.g, is included in the product distribution.

```
#!/usr/cogent/bin/phgamma

/*
This example draws many buttons on the screen.
The PtExtentWidget function is used to make sure
the window can hold all of the widgets.
*/

require_lisp("PhotonWidgets");
PtInit(nil);
win = new(PtWindow);
win.resize_flags = Pt_RESIZE_X_ALWAYS | Pt_RESIZE_Y_ALWAYS;
PtRealizeWidget(win);

but_array = make_array(0);
for(i=0;i<100;i++)
{
  but_array[i] = new(PtButton);
  but_array[i].text_string = "Button";
  but_array[i].SetPos((i * 10) % 640 , (i * 10) % 480);
  PtRealizeWidget(but_array[i]);
  PtExtentWidget(win);
  flush_events();
  nanosleep(0,50000000);
}

for(i=99;i>=0;i --)
{
  PtDestroyWidget(but_array[i]);
  PtExtentWidget(win);
  flush_events();
  nanosleep(0,50000000);
}
PtMainLoop();
```

See Also

In Photon documentation: PtDestroyWidget.

PtFindDisjoint

`PtFindDisjoint` — finds the nearest disjoint parent widget.

Syntax

```
PtFindDisjoint (widget)
```

Arguments

`widget`

Any PtWidget.

Returns

A PtWidget, or `nil`.

Description

A disjoint widget is one that owns one or more regions that are not children of its own parent's regions, such as `PtWindow` and `PtRegion`. This function returns the disjoint parent widget that is nearest to the *`widget`*, which would be the *`widget`* itself if it is disjoint.

See Also

In Photon documentation: PtFindDisjoint.

PtFlush

PtFlush — immediately updates widgets.

Syntax

```
PtFlush ()
```

Arguments

none

Returns

t when successful, else error.

Description

This function makes immediate widget updates and repairs damage.

Example

This example, ex_PtFlush.g, is included in the product distribution.

```
#!/usr/cogent/bin/phgamma

/*
This example draws buttons on the screen, using PtExtentWidget() to
resize the window.  PtFlush() is used to force the drawing of new
buttons before the window is resized.  Commenting out PtFlush() allows
the window to resize as the buttons are created, using flush_events().
*/

require_lisp("PhotonWidgets");
PtInit(nil);
win = new(PtWindow);
win.resize_flags = Pt_RESIZE_X_ALWAYS | Pt_RESIZE_Y_ALWAYS;
PtRealizeWidget(win);

but_array = make_array(0);
for(i=0;i<10;i++)
{
  but_array[i] = new(PtButton);
  but_array[i].text_string = "Button";
  but_array[i].SetPos((i * 20), (i * 15));
  PtRealizeWidget(but_array[i]);
  PtFlush();

  nanosleep(0,500000000);
  PtExtentWidget(win);
  flush_events();
  nanosleep(0,500000000);
}

PtMainLoop();
```

See Also

In Photon documentation: PtFlush.

PtFrameSize

`PtFrameSize` — sets the size of the window frame.

Syntax

`PtFrameSize (render, border_width)`

Arguments

render

The render flags used to define the window.

border_width

The border width of your window in pixels: either 0, or the value of the border_width flag of your window.

Returns

A list in the format (l,t,r,b) which is the widths of the left, top, right and bottom borders, in pixels.

Description

This function uses the *render* information about your window to set border widths for the left, top, right, and bottom sides. The *border_width* can be set to 0, or to the border_width value of your window.

See Also

In Photon documentation: PtFrameSize.

PtForwardWindowEvent

`PtForwardWindowEvent` — forwards a window event to the window manager.

Syntax

```
PtForwardWindowEvent (PtWindowEvent)
```

Arguments

`PtWindowEvent`

A Photon Window event.

Returns

`t` on success, else `nil`.

See Also

In Photon documentation: PtForwardWindowEvent

PtGetAbsPosition

`PtGetAbsPosition` — gives the absolute position of a widget.

Syntax

```
PtGetAbsPosition (widget)
```

Arguments

`widget`

Any widget.

Returns

The value argument, or `nil`.

Description

This function returns the absolute position of the *`widget`*. The returned position is an instance of the PhPoint class.

Example

```
Gamma> PtInit(nil);
t
Gamma> win = new(PtWindow);
window definition
Gamma> PtRealizeWidget(win);
t
Gamma> PtGetAbsPosition(win);
{PhPoint (x . 32) (y . 58)}
```

See Also

In Photon documentation: PtGetAbsPosition.

PtGetParent

PtGetParent — gives the class definition for a parent of a widget.

Syntax

```
PtGetParent (widget, class)
```

Arguments

widget

The widget whose parent is being looked up.

class

The class of the parent.

Returns

The definition of the parent class if successful, else nil.

Example

```
Gamma> PtInit(nil);
t
Gamma> win = new(PtWindow);
window definition
Gamma> pane = new(PtPane);
pane definition
Gamma> a = new(PtLabel);
label definition
Gamma> PtGetParent(a,PtWindow);
window definition
```

See Also

PtReParentWidget, PtSetParentWidget

PtGetParentMember

PtGetParentMember — not yet documented.

Syntax

```
PtGetParentMember (widget, class);
```

Arguments

widget

class

Returns

PtHit

`PtHit` — searches a container for a widget.

Syntax

```
PtHit (container, n, rect)
```

Arguments

container

A PtContainer widget.

n

The widget number to search for.

rect

An area in the form of a PtRect widget.

Returns

A widget on success, else `nil`.

Description

This function searches the *container* for the *n*th widget that intersects the rectangle *rect*. The *rect* will be relative to the basic widget canvas of the *container*. This function ignores unrealized or procreated widgets.

See Also

In Photon documentation: PtHit.

PtHold, PtRelease

PtHold, PtRelease — hold/release the issue of all Photon events to the Photon server.

Syntax

```
PtHold ()
PtRelease ()
```

Arguments

none

Returns

A global hold count value.

Description

PtHold prevents all Photon events from being issued to the Photon server. This is useful when a program makes a large number of changes to either overlapping widgets or widgets contained in the same container. Using PtHold can prevent the container from being re-drawn after each change. Normally, PhEventNext is called within the Gamma next_event function, allowing events to be processed. Using PtHold prevents PhEventNext from being executed within next_event.

PtHold will not stop all Photon events from being submitted to the Photon server indefinitly. Once the application's draw buffer is full, the events will be sent to the Photon server, regardless of PtHold.

Each time PtHold is called, the global hold count increases by one. This hold count is decreased by one each time PtRelease is called. When the global hold count decreases to 0, the held Photon events are submitted to the Photon server through the event loop once again. Thus PtHold and PtRelease are typically called conjointly.

See Also

In Photon documentation: PtHold, PtRelease.

PtInit

`PtInit` — initializes the widget library.

Syntax

`PtInit (`*`name`*`)`

Arguments

`name`

The name of the Photon server to attach to. If `nil` is passed then attach to the local Photon server.

Returns

`t` if the attachment was successful, otherwise `nil`.

Description

This function initializes the widget library. It must be called before any Photon functions are used in your application.

Example

To attach to the local Photon server:

```
Gamma> PtInit (nil);
t
```

To attach to a Photon server running on node 5:

```
Gamma> PtInit ("//5/dev/photon");
t
```

See the introduction of the Photon Functions chapter (Tutorial One) in the Cogent Tools Demo and Tutorials book for an example of this function used in context.

See Also

In Photon documentation: PtInit.

PtInitDrag

`PtInitDrag` — starts a drag operation.

Syntax

`PtInitDrag (`*`widget`*`, `*`container`*`, `*`flags`*`)`

Arguments

`widget`

The widget on which the dragging events are to be enabled.

`container`

The container within which the dragging events are to be confined.

`flags`

Determiners for the drag operation. For details see PhInitDrag in the Photon documentation.

Returns

`nil` on success.

Example

This example, `ex_PtInitDrag.g`, is included in the product distribution.

```
#!/usr/cogent/bin/phgamma

/*
This example puts a small window on the screen with a text label that
can be dragged.
*/

require_lisp("PhotonWidgets");

PtInit(nil);

win = new(PtWindow);
win.SetDim(200,200);
PtRealizeWidget(win);

lab = new(PtLabel);
lab.text_string = "Drag Me";
PtRealizeWidget(lab);

DraggedWidget := nil;

method PtWidget.StartDrag ()
{
   DraggedWidget = self;
   PtInitDrag (self, nil, Ph_TRACK_DRAG | Ph_DRAG_TRACK);
}

function handle_drag ()
{
   local   event = cbinfo.event, rect;

   if (event.type == Ph_EV_DRAG)
   {
      if (event.subtype == Ph_EV_DRAG_COMPLETE ||
```

```
            event.subtype == Ph_EV_DRAG_MOVE)
      {
         if (DraggedWidget)
         {
            rect = TranslateRect (event_data.drag_event.rect,
              event.translation);
            DraggedWidget.SetPos (rect.ul.x, rect.ul.y);
         }
         if (event.subtype == Ph_EV_DRAG_COMPLETE)
            DraggedWidget = nil;
      }
   }
}

PtAttachCallback(win,Pt_CB_RAW,#handle_drag(),Ph_EV_DRAG);
PtAttachCallback(lab,Pt_CB_RAW,#widget.StartDrag(),Ph_EV_BUT_PRESS);

PtMainLoop();
```

See Also

In Photon documentation: PhInitDrag.

PtMainLoop

`PtMainLoop` — starts an infinite event loop.

Syntax

```
PtMainLoop ()
```

Arguments

none

Returns

None. Never returns.

Description

This function starts an infinite event loop. If this function is read from a file it closes the file before starting the event loop. This is useful to ensure that no files are left open before starting the main event loop.

Example

See examples for PtAttachCallback, PtContainerGiveFocus, and PtExtentWidget.

See the Photon: the Controller window of the Controller Functions chapter (Tutorial Two) in the Cogent Tools Demo and Tutorials book for an example of this function used in context.

See Also

`next_event`, `next_event_nb`

and in Photon documentation: PtMainLoop.

PtModalStart, PtModalEnd

PtModalStart, PtModalEnd — start and end modal processing.

Syntax

```
PtModalStart ()
PtModalEnd (start_number)
```

Arguments

start_number

The modal-processing loop number returned from a call to PtModalStart.

Returns

PtModalStart returns an integer used as the start-number for PtModalEnd.

PtModalEnd returns t if modal processing was successfully exited.

Description

PtModalStart initiates modal processing, which halts the current event loop and starts a new loop. This is useful for creating a dialog box where the user must make some input before accessing any other widget in the interface. To block and unblock other windows, you have to use the Pt_BLOCKED resource, as demonstrated in the example below. When the dialog or other use of modal processing is finished, the modal event loop is exited by calling PtModalEnd.

Example

This example, ex_PtModalStart.g, is included in the product distribution.

```
#!/usr/cogent/bin/phgamma

/*
This example creates a window with a label you can drag, and a
"Freeze" button.  Pressing the "Freeze" button initiates
PtModalStart() and opens another window, with an "Unfreeze" button.
Until the "Unfreeze" button is pressed, the first window is frozen.
*/

require_lisp("PhotonWidgets");

PtInit(nil);

//The main window.

win = new(PtWindow);
win.SetArea(300,50,250,250);
PtRealizeWidget(win);

lab = new(PtLabel);
lab.text_string = "Drag Me";
lab.SetPos(90, 30);
PtRealizeWidget(lab);

button1 = new(PtButton);
button1.text_string = "Freeze label and open message";
button1.SetPos(30, 75);
PtAttachCallback(button1, Pt_CB_ACTIVATE, #doOpen());
```

```
PtRealizeWidget(button1);

exitbut = new(PtButton);
exitbut.SetPos(110, 175);
exitbut.text_string = "Exit";
PtAttachCallback(exitbut,Pt_CB_ACTIVATE,#exit_program(1));
PtRealizeWidget(exitbut);

//Functions to start and end modal state.

function PhabModalLoop(done)
{
 local count = nil;
 protect
 {
  count = PtModalStart();
  while(!eval(done))
   PtProcessEvent();
 }
 unwind
 {
  if(count != nil)
   PtModalEnd(count);
 }
}

function doOpen()
{
  make_msg();
  done = nil;
  win.flags = Pt_BLOCKED;
  PhabModalLoop(eval(done));
}

function doClose()
{
 done = t;
 win.flags = cons(Pt_BLOCKED, nil);
}

//The modal_state window.

/* This function makes the modal_state window using a PtMessage widget.*/
function make_msg()
{
  win2 = new(PtMessage);
  win2.msg_text = "Press Cancel to continue.\nPress Print to print this\nmessage and continue.\n ";
  win2.msg_title = "The message.";
  win2.flags = Pt_MSG_CENTER_ON_PARENT;
  win2.msg_button1 = "Cancel";
  win2.msg_button2 = "Print";

  PtAttachCallback(win2, Pt_CB_MSG_BUTTON2, 'princ(@win2.msg_text, "\n"));
  PtAttachCallback(win2, Pt_CB_MSG_BUTTON2, #doClose());
  PtAttachCallback(win2, Pt_CB_MSG_BUTTON1, #doClose());
  PtRealizeWidget(win2);
}

/*  An alternate way to make the window, using a PtWindow widget.
 If used, the following lines must be added...
 to doOpen()  :  PtRealizeWidget(win2);
 to doClose() :  PtUnrealizeWidget(win2);
 and the line :  make_msg();
 should be removed from doOpen().

win2 = new(PtWindow);
win2.SetArea(440,100,100,80);
lab2 = new(PtLabel);
```

```
lab2.text_string = "Press button to continue.";
button2 = new(PtButton);
button2.text_string = "Unfreeze";
button2.SetPos(40, 30);
PtAttachCallback(button2, Pt_CB_ACTIVATE, #doClose());
*/

//Functions to create a drag_able widget.

DraggedWidget := nil;

method PtWidget.StartDrag ()
{
   DraggedWidget = self;
   PtInitDrag (self, nil, Ph_TRACK_DRAG | Ph_DRAG_TRACK);
}

function handle_drag ()
{
   local   event = cbinfo.event, rect;

   if (event.type == Ph_EV_DRAG)
   {
      if (event.subtype == Ph_EV_DRAG_COMPLETE ||
            event.subtype == Ph_EV_DRAG_MOVE)
      {
         if (DraggedWidget)
         {
            rect = TranslateRect (event_data.drag_event.rect,
              event.translation);
            DraggedWidget.SetPos (rect.ul.x, rect.ul.y);
         }
         if (event.subtype == Ph_EV_DRAG_COMPLETE)
            DraggedWidget = nil;
      }
   }
}

PtAttachCallback(win,Pt_CB_RAW,#handle_drag(),Ph_EV_DRAG);
PtAttachCallback(lab,Pt_CB_RAW,#widget.StartDrag(),Ph_EV_BUT_PRESS);

PtMainLoop();
```

See Also

In Photon documentation: PtModalStart and PtModalEnd.

PtProcessEvent

`PtProcessEvent` — handles Photon events.

Syntax

```
PtProcessEvent ()
```

Arguments

none

Returns

`t` if an event was processed. Otherwise, it doesn't return, and puts the application into reply-blocked mode.

See Also

In Photon documentation: PtProcessEvent

PtProtectCallbacks

`PtProtectCallbacks` — protects code from callback errors.

Syntax

```
PtProtectCallbacks (state)
```

Arguments

`state`

The protection state: `t` or `nil`.

Returns

`1` if the *`state`* is `t`, `0` if it is `nil`.

Description

This function is designed to be used in code that surrounds callbacks. As such, it protects code related to those callbacks from possible errors. If, for example, an error occurs, the error will be thrown but the callback function will complete safely. This means that if more than one error occurs in processing a callback, only the first error will be reported.

Of course, if this function is *not* called and errors occur, they will all be reported. However, the callback will terminate early, which could cause failures in subsequent Photon calls. This is especially noticeable if the error causes the drawing state to get damaged, so that subsequent draws do not complete properly. You will find areas of the screen are not redrawn, or are redrawn with the wrong data. The effect is localized to your program, though, so it won't harm the underlying Photon environment.

PtQueryCallbacks

PtQueryCallbacks — finds widget callback functions.

Syntax

```
PtQueryCallbacks (widget, callback_number)
```

Arguments

widget

Any widget.

callback_number

A callback-number or symbol equating to a callback-number.

Returns

A list of all functions associated with the callback.

Example

This example, ex_PtQueryCallbacks.g, is included in the product distribution.

```
#!/usr/cogent/bin/phgamma

/*
This code puts up a window with a button.  Pressing the button creates
two text boxes.  The second box displays the return value of the
PtQueryCallbacks() call. Also, each time the button is pressed, the
return value is printed.
*/

require_lisp("PhotonWidgets");
PtInit(nil);

win = new(PtWindow);
win.SetDim(200,200);
PtRealizeWidget(win);

button = new(PtButton);
button.text_string = "Press Here.";
button.SetPos(60,20);
PtAttachCallback(button, Pt_CB_ACTIVATE, #title());
PtAttachCallback(button, Pt_CB_ACTIVATE, #functionlist());
PtRealizeWidget(button);

function title()
{
 txt = new(PtText);
 txt.SetPos(30,80);
 txt.SetDim(150,25);
 a = PtQueryCallbacks(button, Pt_CB_ACTIVATE);
 txt.text_string = string("The callback functions:");
 PtRealizeWidget(txt);
}

function functionlist()
{
 txt = new(PtText);
 txt.SetPos(30,115);
 txt.SetDim(150,25);
```

```
 c = PtQueryCallbacks(button, Pt_CB_ACTIVATE);
 txt.text_string = string(c);
 princ(c,"\n");
 PtRealizeWidget(txt);
}

PtMainLoop();
```

PtRawSetPos

`PtRawSetPos` — is obsolete.

Syntax

```
PtRawSetPos (widget, pos)
```

Description

This function is obsolete and scheduled for removal.

PtReParentWidget

PtReParentWidget — changes parents of widgets.

Syntax

PtReParentWidget (*widget*, *parent_widget*)

Arguments

widget

The widget to re-parent.

parent_widget

The new parent of the *widget*.

Returns

t when successful, otherwise an error message.

Description

This function takes the *widget* from its current parent and associates it with the specified *parent_widget*.

Example

In this example a PtWindow instance is created, which is a container. Once this container is created, it is the default parent until anther container is created. A label is created, then another container (a PtPane), and then another label. The first label (a) is a child of of the PtWindow. The second label (b) is a child of the PtPane, since it is created while the pane is the default container.

The PtGetParent call shows that the second PtLabel (b) has both pane and win for parents. The second label is then re-parented to the PtWindow container. The PtGetParent call now shows that the second PtLabel (b) has only win as a parent.

```
Gamma> PtInit(nil);
t
Gamma> win = new(PtWindow);
window definition
Gamma> a = new(PtLabel);
label definition
Gamma> pane = new(PtPane);
pane definition
Gamma> b = new(PtLabel);
label definition
Gamma> PtGetParent(b,PtPane) == pane;
t
Gamma> PtGetParent(b,PtWindow) == win;
t
Gamma> PtReParentWidget(b,win);
t
Gamma> PtGetParent(b,PtPane) == pane;
nil
Gamma> PtGetParent(b,PtWindow) == win;
t
Gamma>
```

This example, `ex_PtSet-ReParentWidget.g`, is included in the product distribution.

```
#!/usr/cogent/bin/phgamma

/*
This example demonstrates the PtReParentWidget and PtSetParentWidget
functions.  It creates a window with an embedded pane, and puts a label
and a button inside the pane. The default parent of both the label and
the button is the pane.

Toggling the button activates PtReParentWidget for the label, causing
the label to move from the pane to the original window, and back.

Uncommenting the PtSetParentWidget() call sets the window as the parent
widget of the label, so when the program is restarted the label appears
in the upper left corner of the window.
*/

function switch_parents(button, label, p1, p2)
{
  if ((button.flags & Pt_SET) != 0)
 PtReParentWidget (label, p1);
  else
 PtReParentWidget (label, p2);
  PtRealizeWidget(p1);
}

require_lisp("PhotonWidgets");
PtInit(nil);

win = new(PtWindow);

pane = new(PtPane);
pane.fill_color = PgRGB(160,160,160);
pane.SetArea(100,0,100,100);

//PtSetParentWidget (win);

label = new(PtLabel);
label.text_string = "Label";

but = new(PtButton);
but.SetArea(25, 40, 50, 40);
but.text_string = "Switch\nParent";
but.flags = Pt_TOGGLE;
PtAttachCallback(but, Pt_CB_ACTIVATE, ‘switch_parents(@but, @label, @win, @pane));

PtRealizeWidget(win);
PtMainLoop();
```

See Also

PtGetParent, PtSetParentWidget

and in Photon documentation: PtReParentWidget.

PtRealizeWidget

`PtRealizeWidget` — initializes widgets.

Syntax

`PtRealizeWidget (`*`widget`*`)`

Arguments

`widget`

The widget to realize or re-realize.

Returns

`t` if successful, otherwise `nil`.

Description

This function initializes a widget. The widget's children are automatically realized as well if the widget is a container-type widget and the `Pt_DELAY_REALIZE` or `Pt_DELAY_ACTIVATION` flags are not set. Once a widget is realized it is visible and may be interactive.

Example

For another example, see PtUnrealizeWidget.

```
Gamma> PtInit (nil);
t
Gamma> win = new(PtWindow);
window instance
Gamma> PtRealizeWidget (win);
t
```

See Also

PtUnrealizeWidget

and in Photon documentation: PtRealizeWidget.

PtRemoveCallback

PtRemoveCallback — removes a callback.

Syntax

```
PtRemoveCallback (widget, callback_number, function)
```

Arguments

widget

The widget to remove the callback function from.

callback_number

The callback type.

function

The function to remove.

Returns

t when successful.

Example

This example, ex_PtRemoveCallback.g, is included in the product distribution.

```
#!/usr/cogent/bin/phgamma

/*
This example attaches a callback to a button to start pfm (Photon File
Manager).  After 10 seconds the callback is removed and the text on
the face of the button is changed.
*/

function start_pfm ()
{
   system("pfm &");
}

function remove_callback (button)
{
   button.text_string = "I no longer start PFM";
   PtRemoveCallback(but,Pt_CB_ACTIVATE,code);
}

PtInit(nil);
win = new(PtWindow);
but = new(PtButton);
but.text_string = "Press Me to Start PFM";
PtRealizeWidget(win);

code = #start_pfm();

PtAttachCallback(but,Pt_CB_ACTIVATE,code);
after(10,#remove_callback(but));

PtMainLoop();
```

See Also

PtAttachCallback

PtResourceName

PtResourceName — finds a resource name from a resource number.

Syntax

PtResourceName (*widget*|*class*, *resource_number*)

Arguments

widget|*class*

The name of a widget or class.

resource_number

The resource number of the widget or class whose name you want to find.

Returns

The name of the resource.

Example

```
Gamma> PtInit(nil);
t
Gamma> require_lisp("PhabTemplate");
"//4/usr/cogent/lib/PhabTemplate.lsp"
Gamma> PtResourceName(PtPane, 2006);
top_border_color
Gamma> PtResourceName(PtButton, 3011);
text_string
Gamma> PtResourceName(PtWindow, 1008);
resize_flags
Gamma>
```

PtSetParentWidget

PtSetParentWidget — turns a widget into a parent widget.

Syntax

```
PtSetParentWidget (widget)
```

Arguments

widget

The container-type widget that will become the new parent widget.

Returns

If successful, the new parent widget, otherwise nil.

Example

This example, ex_PtSet-ReParentWidget.g, is included in the product distribution.

```
#!/usr/cogent/bin/phgamma

/*
This example demonstrates the PtReParentWidget and PtSetParentWidget
functions.  It creates a window with an embedded pane, and puts a label
and a button inside the pane. The default parent of both the label and
the button is the pane.

Toggling the button activates PtReParentWidget for the label, causing
the label to move from the pane to the original window, and back.

Uncommenting the PtSetParentWidget() call sets the window as the parent
widget of the label, so when the program is restarted the label appears
in the upper left corner of the window.
*/

function switch_parents(button, label, p1, p2)
{
  if ((button.flags & Pt_SET) != 0)
 PtReParentWidget (label, p1);
  else
 PtReParentWidget (label, p2);
  PtRealizeWidget(p1);
}

require_lisp("PhotonWidgets");
PtInit(nil);

win = new(PtWindow);

pane = new(PtPane);
pane.fill_color = PgRGB(160,160,160);
pane.SetArea(100,0,100,100);

//PtSetParentWidget (win);

label = new(PtLabel);
label.text_string = "Label";

but = new(PtButton);
but.SetArea(25, 40, 50, 40);
```

```
but.text_string = "Switch\nParent";
but.flags = Pt_TOGGLE;
PtAttachCallback(but, Pt_CB_ACTIVATE, ‘switch_parents(@but, @label, @win, @pane));

PtRealizeWidget(win);
PtMainLoop();
```

See Also

In Photon documentation: PtSetParentWidget.

PtSyncPhoton

PtSyncPhoton — flushes the current draw buffer.

Syntax

```
PtSyncPhoton ()
```

Arguments

none

Returns

t.

Description

This function flushes the current draw buffer, performing all pending widget updates and removals. These operations are usually performed automatically, but some applications that modify or destroy widgets may need to call PtSyncPhoton explicitly. This function should never be used in the callback of a widget that has been destroyed.

See Also

In Photon documentation: PtSyncPhoton.

PtTranslateRect

PtTranslateRect — changes the position of PhRect class instances.

Syntax

TranslateRect (*original_rect*, *translation_point*)

Arguments

original_rect

An instance of a PhRect class.

translation_rect

An instance of a PhPoint class.

Returns

The *original_rect* instance of the PhRect class with its members modified by the amount of the *translation_point*.

Description

This function allows users to create instances of the PhPoint class to act as modifiers to PhRect instances. Mathematically, the result of the function (result), in relation to the original PhRect instance (original) and the translation (point) is:

- **result.lr.x** = original.lr.x + point.x
- **result.lr.y** = original.lr.y + point.y
- **result.ul.x** = original.ul.x + point.x
- **result.ul.y** = original.ul.y + point.y

Example

This output:

```
A rectangle:
{PhRect (lr . {PhPoint (x . 7) (y . 9)})
        (ul . {PhPoint (x . 4) (y . 1)})}

A translation point:
{PhPoint (x . 20) (y . 30)}

The rectangle, translated:
{PhRect (lr . {PhPoint (x . 27) (y . 39)})
        (ul . {PhPoint (x . 24) (y . 31)})}
```

Was generated using this code:

```
#!/usr/cogent/bin/phgamma

require_lisp("PhotonWidgets");
PtInit(nil);

r1 = new(PhPoint);
r2 = new(PhPoint);
TransPoint = new(PhPoint);
```

```
r1.x = 4;
r1.y = 1;
r2.x = 7;
r2.y = 9;

TransPoint.x = 20;
TransPoint.y = 30;

RectOrig = new(PhRect);
RectOrig.ul = r1;
RectOrig.lr = r2;
princ("A rectangle:\n",RectOrig, "\n\n");

princ("A translation point:\n",TransPoint, "\n\n");

TransRect = PtTranslateRect(RectOrig, TransPoint);
princ("The rectangle, translated:\n", TransRect, "\n\n");
```

See Also

In Photon documentation: PtTranslateRect.

PtUnrealizeWidget

`PtUnrealizeWidget` — hides a widget.

Syntax

```
PtUnrealizeWidget (widget)
```

Arguments

`widget`

The widget to unrealize.

Returns

If successful the unrealized widget is returned, otherwise `nil`.

Description

This function makes a widget disappear. Actions associated with the widget are no longer available to the user. This function does not destroy the widget, it only hides it.

Example

This example, `ex_PtUnrealizeWidget.g`, is included in the product distribution.

```
#!/usr/cogent/bin/phgamma

/*
This example creates a small window with a button in it.  The every()
and at() functions are used to create timers to realize and unrealize
the button every 2 seconds, and change the label correspondingly.
*/

require_lisp("PhotonWidgets");
PtInit(nil);
win = new(PtWindow);
win.SetDim(220, 150);

b = new(PtButton);
b.text_string = "The button is visible";
b.SetPos(40, 60);

label1 = new(PtLabel);
label1.SetPos(20,15);
label1.text_string = string("The button will realize and\n",
       "unrealize every 2 seconds.");

label2 = new(PtLabel);
label2.SetPos(70,100);
label2.text_string = "Realized";

PtExtentWidget(win);
PtRealizeWidget(win);

function realize_wgt(wgt, label, secs)
{
  PtRealizeWidget(wgt);
  label.text_string = "Realized";
  after(secs, 'unrealize_wgt(@wgt, @label));
}
```

```
function unrealize_wgt(wgt, label)
{
  PtUnrealizeWidget(wgt);
  label.text_string = "Unrealized";
}

after(2, ‘unrealize_wgt(b, label2));
every(4,#realize_wgt(b, label2, 2));

PtMainLoop();
```

See Also

PtRealizeWidget, PtDestroyWidget

and in Photon documentation: PtUnrealizeWidget.

PtUpdate

`PtUpdate` — decrements the global hold count for widget repair.

Syntax

```
PtUpdate ()
```

Arguments

none

Returns

The current hold count if successful, otherwise `nil`.

Description

This function decrements the global hold count. When this count reaches 0, all widget damage is repaired and all widget visibility flags (such as `Pt_OBSCURED` and `Pt_CLEAR`) are updated. This function differs from `PtRelease`, which decrements the global *widget* hold count.

See Also

PtHold, PtRelease

and in Photon documentation: PtUpdate

PtWidgetChildren

PtWidgetChildren — finds the children of a widget.

Syntax

```
PtWidgetChildren (widget)
```

Arguments

widget

The widget whose children are to be searched for.

Returns

A list of child widgets, if there are any, including definitions. Otherwise, nil.

Description

Example

This example, ex_PtWidgetChildren.g, is included in the product distribution.

```
#!/usr/cogent/bin/phgamma

/*
The following example makes a window with a pane and a label, and
prints the output of PtWidgetChildren() for the pane widget.  It
then prints the function output for the label widget, which is nil, as
it has no children.
*/

require_lisp("PhotonWidgets");
PtInit(nil);

win = new(PtWindow);
win.SetDim(150,150);
/*
but = new(PtButton);
but.text_string = "A button.";
*/
pane = new(PtPane);
pane.SetArea(20,20,100,100);
pane.fill_color = PgRGB(160,190,160);

label = new(PtLabel);
label.SetPos(25,25);
label.text_string = "A label.";


PtRealizeWidget(win);
pretty_princ("\nThe children of the ", class_name(pane), " are:\n",
    PtWidgetChildren(pane),"\n");
pretty_princ("\nThe children of the ", class_name(label), " are:\n",
    PtWidgetChildren(label),"\n");

PtMainLoop();
```

PtWidgetParent

`PtWidgetParent` — finds the parent of a widget.

Syntax

`PtWidgetParent (`*`widget`*`)`

Arguments

`widget`

Any widget.

Returns

The parent widget, if one exists, otherwise `nil`.

Example

```
Gamma> PtInit(nil);
t
Gamma> win = new(PtWindow);
window instance
Gamma> pane = new(PtPane);
pane instance
Gamma> label = new(PtLabel);
label intance
Gamma> PtWidgetParent(label) == pane;
t
Gamma> PtWidgetParent(pane) == win;
t
Gamma> PtWidgetParent(win);
nil
```

PtWidgetToBack, PtWidgetToFront

PtWidgetToBack, PtWidgetToFront — moves a widget behind or in front of others.

Syntax

```
PtWidgetToBack (widget)
PtWidgetToFront (widget)
```

Arguments

widget

The widget to move.

Returns

t if successful, otherwise nil.

Description

These functions move the *widget*, and any of its children, behind or in front of all of the other widgets that are inside the same container.

Example

This example, ex_PtWidgetToBack.g, is included in the product distribution.

```
#!/usr/cogent/bin/phgamma

/*
This example creates a window with three colored rectangles and three
buttons that move them to front and back.
*/

require_lisp("PhotonWidgets");
PtInit(nil);

win = new(PtWindow);
win.SetDim(300,200);

RectA = new(PtRect);
RectA.fill_color = PgRGB(255,0,0);
RectA.SetPos(140,45);
RectA.SetDim(100,80);

RectB = new(PtRect);
RectB.fill_color = PgRGB(0,150,120);
RectB.SetPos(70,60);
RectB.SetDim(80,70);

RectC = new(PtRect);
RectC.fill_color = PgRGB(0,30,220);
RectC.SetPos(110,20);
RectC.SetDim(90,90);

butA = new(PtButton);
butB = new(PtButton);
butC = new(PtButton);

butA.text_string = "Red";
butB.text_string = "Green";
```

```
butC.text_string = "Blue";

butA.SetPos(55,100);
butB.SetPos(115,100);
butC.SetPos(175,100);

Instructions = new(PtLabel);
Instructions.text_string = "Hold moves to front, release moves to back.";
Instructions.SetPos(10,150);

PtAttachCallback(butA, Pt_CB_ARM, #PtWidgetToFront(RectA));
PtAttachCallback(butB, Pt_CB_ARM, #PtWidgetToFront(RectB));
PtAttachCallback(butC, Pt_CB_ARM, #PtWidgetToFront(RectC));

PtAttachCallback(butA, Pt_CB_DISARM, #PtWidgetToBack(RectA));
PtAttachCallback(butB, Pt_CB_DISARM, #PtWidgetToBack(RectB));
PtAttachCallback(butC, Pt_CB_DISARM, #PtWidgetToBack(RectC));

PtRealizeWidget(win);
PtMainLoop();
```

See Also

PtRealizeWidget, PtUnrealizeWidget

TranslateRect

`TranslateRect` — is obsolete. See PtTranslateRect.

Index

I

P

T

Colophon

This book was produced by Cogent Real-Time Systems, Inc. from a single-source group of SGML files. Gnu Emacs was used to edit the SGML files. The DocBook DTD and related DSSSL stylesheets were used to transform the SGML source into HTML, PDF, and QNX Helpviewer output formats. This processing was accomplished with the help of OpenJade, JadeTeX, Tex, and various scripts and makefiles. Details of the process are described in our book: Preparing Cogent Documentation, which is published on-line at
`http://developers.cogentrts.com/cogent/prepdoc/book1.html`.

Text written by Bob McIlvride.