Documentation Library

Gamma/Photon Reference Volume 2: Classes

Using Gamma™ with the Photon microGUI®, Version 1.1

Cogent Real-Time Systems, Inc.

August 11, 2010

Gamma/Photon Reference Volume 2: Classes: 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

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. Widget Classes

Table of Contents

CwGraph

CwGraph — A Cogent graph widget.

Synopsis

```
class CwGraph PtBasic
{
    graph_flags;                    // flag
    graph_traces;                   // long
    graph_x_axis_color;             // color
    graph_x_axis_width;             // integer
    graph_x_font;                   // string
    graph_x_label;                  // string
    graph_x_label_color;            // color
    graph_x_major_length;           // integer
    graph_x_major_tick;             // double array
    graph_x_minor_length;           // integer
    graph_x_minor_tick;             // double array
    graph_xmax;                     // double array
    graph_xmin;                     // double array
    graph_y_axis_color_left;        // color
    graph_y_axis_color_right;       // color
    graph_y_axis_width_left;        // integer
    graph_y_axis_width_right;       // integer
    graph_y_font_left;              // string
    graph_y_font_right;             // string
    graph_y_label_color_left;       // color
    graph_y_label_color_right;      // color
    graph_y_label_left;             // string
    graph_y_label_right;            // string
    graph_y_major_length_left;      // integer
    graph_y_major_length_right;     // integer
    graph_y_major_tick_left;        // double array
    graph_y_major_tick_right;       // double array
    graph_y_minor_length_left;      // integer
    graph_y_minor_length_right;     // integer
    graph_y_minor_tick_left;        // double array
    graph_y_minor_tick_right;       // double array
    graph_ymax_left;                // double array
    graph_ymax_right;               // double array
    graph_ymin_left;                // double array
    graph_ymin_right;               // double array
}
```

Base Classes

PtWidget <-- PtBasic <-- CwGraph

Description

This widget is a graph that allows plotting multiple curves of arbitrary length, with options for one x-axis display, and two independent y-axis displays (left and right). It uses floating-point numbers in any scale, with independent scaling for each curve.

This widget does not appear in the Photon documentation.

Instance Variables

graph_flags

This instance variable may be a combination of zero or more of the following flags. Most of these flags correspond to instance variables in the widget. Unsetting a flag here overrides the assignment of any corresponding instance variables.

Constant	Description
Cw_GRAPH_X_AXIS	Show the x-axis on the graph. Default is set.
Cw_GRAPH_X_AXIS_DATE	Not currently implemented.
Cw_GRAPH_X_AXIS_LABEL	Show numbers on the x-axis. Default is set.
Cw_GRAPH_X_MAJOR_TICK	Show major tick marks on the x-axis. Default is set. Unsetting this variable will make the major tick marks appear as minor tick marks if Cw_GRAPH_X_MINOR_TICK is set, otherwise the major tick marks will not appear at all.
Cw_GRAPH_X_MINOR_TICK	Show minor tick marks on the x-axis. Default is set.
Cw_GRAPH_X_AXIS_TIME	Show the `graph_major_ticks` on the x-axis label as times. Default is not set.
Cw_GRAPH_X_AXIS_TITLE	Show the `graph_x_label` string on the x-axis label. Default is set.
Cw_GRAPH_Y_AXIS_LEFT, Cw_GRAPH_Y_AXIS_RIGHT	Show the left or right y-axis on the graph. Default is set.
Cw_GRAPH_Y_AXIS_LABEL_LEFT, Cw_GRAPH_Y_AXIS_LABEL_RIGHT	Show numbers on the left or right y-axis. Default is set.
Cw_GRAPH_Y_MAJOR_TICK_LEFT, Cw_GRAPH_Y_MAJOR_TICK_RIGHT	Show major tick marks on the left or right y-axis. Default is set. Unsetting this variable will make the major tick marks appear as minor tick marks if either of the Cw_GRAPH_Y_MINOR_TICK flags are set, otherwise the major tick marks will not appear at all.
Cw_GRAPH_Y_MINOR_TICK_LEFT, Cw_GRAPH_Y_MINOR_TICK_RIGHT	Show minor tick marks on the left or right y-axis. Default is set.
Cw_GRAPH_Y_AXIS_TITLE_LEFT, Cw_GRAPH_Y_AXIS_TITLE_RIGHT	Show the `graph_x_label` string on the left or right y-axis label. Default is set.

`graph_traces`

A positive integer specifying how many traces the widget will use. The traces are identified by consecutive integers starting with 0.

`graph_x_axis_color`

A number specifying the color of the x-axis. Default is `0x0` (black).

`graph_x_axis_width`

An integer specifying the width in pixels of the x-axis line. Default is `1`.

`graph_x_font`

A string specifying the font for the x-axis display. Default is `"helv12"`.

`graph_x_label`

A string specifying the label to use for the x-axis. Default is `"X"`.

`graph_x_label_color`

A number specifying the color to use for the x-axis line. Default is `0x0` (black).

`graph_x_major_length`

An integer specifying the length of the major tick marks of the x-axis, in pixels. Default is `5`.

`graph_x_major_tick`

A number specifying the interval of the major tick marks on the x-axis.

`graph_x_minor_length`

An integer specifying the length of the minor tick marks of the x-axis, in pixels. Default is `2`.

`graph_x_minor_tick`

A number specifying the interval of the minor tick marks on the x-axis.

`graph_xmax`

A number specifying the maximum value on the x-axis. This number determines the upper limit of x values to be displayed.

`graph_xmin`

A number specifying the minimum value on the x-axis. This number determines the lower limit of x values to be displayed.

`graph_y_axis_color_left`, `graph_y_axis_color_right`

A number specifying the color of the left or right y-axis. Default is `0x0` (black).

`graph_y_axis_width_left`, `graph_y_axis_width_right`

An integer specifying the width in pixels of the left or right y-axis line. Default is `1`.

`graph_y_font_left`, `graph_y_font_right`

A string specifying the font for the left or right y-axis display. Default is `"helv12"`.

`graph_y_label_color_left`, `graph_y_label_color_right`

A number specifying the color to use for the left or right y-axis line. Default is `0x0` (black).

`graph_y_label_left`, `graph_y_label_right`

A string specifying the label to use for the left or right y-axis. Default is `"Y"`.

`graph_y_major_length_left`, `graph_y_major_length_right`

An integer specifying the length of the major tick marks of the left or right y-axis, in pixels. Default is `5`.

`graph_y_major_tick_left`, `graph_y_major_tick_right`

A number specifying the interval of the major tick marks on the left or right y-axis.

`graph_y_minor_length_left`, `graph_y_minor_length_right`

An integer specifying the length of the minor tick marks of the left or right y-axis, in pixels. Default is `2`.

`graph_y_minor_tick_left`, `graph_y_minor_tick_right`

A number specifying the interval of the minor tick marks on the left or right y-axis.

`graph_ymax_left`, `graph_ymax_right`

A number specifying the maximum value on the left or right y-axis. This number does *not* determine the upper limit of y values to be displayed. That value is determined by the `CwGraphSetYLimits` function.

`graph_ymin_left`, `graph_ymin_right`

A number specifying the minimum value on the left or right y-axis. This number does *not* determine the lower limit of y values to be displayed. That value is determined by the `CwGraphSetYLimits` function.

Convenience Functions

Arguments

These arguments are used in the functions listed below.

buffer	A buffer containing pairs of doubles in binary form, used by `CwGraphAddRawPoints`.
color	A number specifying a color.
count	The number of pairs to read, used by `CwGraphAddRawPoints`.
npoints	A number of points.
onoff	An integer: `0` or `nil` = disable, all other values = enable. Used by `CwGraphEnableTrace`.
pointno	The number of the point in the trace. Points are numbered by consecutive integers, starting at `0`.
points	A number specifying the number of points desired.
start	The starting point for reading the buffer, used by `CwGraphAddRawPoints`.
trace	The trace number.
widget	A `CwGraph` widget.
x	A list of the x coordinates of the *trace*.
y	A list of the y coordinates of the *trace*
ymax	A number specifying the upper limit of y values to be displayed.
ymin	A number specifying the lower limit of y values to be displayed.

Functions

`CwGraphAddRawPoints` `(widget, trace, buffer, start, count)` -- adds points as pairs of double-precision 8-bit numbers.

Returns `t` on success, else error.

`CwGraphAddXYPoints` `(widget, trace, x, y)` -- adds multiple points, using lists of x and y coordinates.

Returns `t` on success, else error.

`CwGraphClearTrace` `(widget, trace)` -- removes a trace.

Returns `t` on success, else error.

`CwGraphEnableTrace` `(widget, trace, onoff)` -- hides or displays a trace on the graph.

Returns `t` on success, else error.

`CwGraphGetScreenData` `(widget, trace, pointno, npoints)` -- gets screen data about specified points in the trace, expressed as a pointer to a binary file.

Returns a pointer to a binary file, or error.

`CwGraphGetTraceData` `(widget, trace, pointno, npoints)` -- gets data about specified points in the trace, expressed as a pointer to a binary file.

Returns a pointer to a binary file.

`CwGraphGetTraceLength` `(widget, trace)` -- gets the total number of points in the *`trace`*.

Returns `t` on success, else error.

`CwGraphSetTraceColor` `(widget, trace, color)` -- sets the color of a trace.

Returns `t` on success, else error.

`CwGraphSetYLimits` `(widget, trace, ymin, ymax)` -- sets the upper and lower y-axis limits for the display of traces. These limits are independent of the `graph_ymin` and `graph_ymax` instance variable values for the widget.

Returns `t` on success, else error.

Example

This example, `ex_CwGraph.g`, included in the product distribution.

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

/*
 * This example demonstrates a CwGraph.
 */

/*
 * Load the required files, which include two dynamic
 * libraries that are not generally needed for Photon
 * widgets.
 */
if (_os_ == "QNX4")
{
  dyna_add_lib("/usr/cogent/lib/photon_s.dlb");
  dyna_add_lib("/usr/cogent/lib/phwidgets.dlb");
}
require_lisp("PhotonWidgets.lsp");
PtInit(nil);

win = new(PtWindow);
grf = new(CwGraph);

grf.SetDim(400,300);
grf.fill_color = 0xffeeaa;

/*
 * Define the x_axis display.
 */
grf.graph_traces = 3;
grf.graph_x_axis_color = 0x00aaff;
grf.graph_x_axis_width = 2;
grf.graph_x_font = "lu10";
grf.graph_x_label = "X Axis";
grf.graph_x_label_color = 0x0;
```

```
grf.graph_x_major_length = 5;
grf.graph_x_major_tick = 20;
grf.graph_x_minor_length = 1;
grf.graph_x_minor_tick = 5;
grf.graph_xmax = 50;
grf.graph_xmin = -50;

/*
 * Define the left y_axis display.  Ignoring the right
 * y_axis will display its default values.
 */
grf.graph_y_axis_color_left = 0x00bbaa;
grf.graph_y_axis_width_left = 2;
//grf.graph_y_font_left = "lu12";
grf.graph_y_font_left = "TextFont10";
grf.graph_y_label_color_left = 0x0;
grf.graph_y_label_left = "Left";
grf.graph_y_major_length_left = 3;
grf.graph_y_major_tick_left = 10.3;
grf.graph_y_minor_length_left = 1;
grf.graph_y_minor_tick_left = 5.3;
grf.graph_ymax_left = 125.3;
grf.graph_ymin_left = -5;

/*
 * Set up the traces.  Each trace has its own
 * y_axis limits, color, and points.
 */
CwGraphSetYLimits(grf,0,-10,10);
CwGraphSetTraceColor(grf,0,0x0);
CwGraphAddXYPoints(grf,0,list(0,13.7,155),list(0,1.3,30));

CwGraphSetYLimits(grf,1,-10,10);
CwGraphSetTraceColor(grf,1,0xaa00aa);
CwGraphAddXYPoints(grf,1,list(-45,-30,0,30,45),list(0,-7,-9,-7,0));

CwGraphSetYLimits(grf,2,-10,10);
CwGraphSetTraceColor(grf,2,0x0000ff);
CwGraphAddXYPoints(grf,2,list(0,20,45,0,-25,-10,0),list(-1.5,-2.5,2.5,3,1.5,0,-1.5));

/*
 * Print the results of various functions as examples.
 */
princ("Trace 0 data: ",CwGraphGetTraceData(grf,0,1,1),"\n");
princ("Trace 0 screen data: ",CwGraphGetScreenData(grf,0,1,1),"\n");
princ("Trace 0 length: ",CwGraphGetTraceLength(grf,0),"\n");
princ("Trace 1 length: ",CwGraphGetTraceLength(grf,1),"\n");
princ("Trace 2 length: ",CwGraphGetTraceLength(grf,2),"\n");

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

Also see A CwGraph Rotating Cube in the Sample Code and Cool Stuff chapter of the Programmer's Manual for a more sophisticated example.

CwMatrix

CwMatrix — A Cogent matrix widget.

Synopsis

```
class CwMatrix PtContainer
{
    matrix_cols;       // short
    matrix_flags;      // flag
    matrix_offset;     // PhPoint
    matrix_range;      // PhRect
    matrix_rows;       // short
    matrix_visible;    // PhRect
}
```

Base Classes

PtWidget <-- PtBasic <-- PtContainer <-- CwMatrix

Description

This widget is the basis for building spreadsheets. It consists of a group of cells arranged in rows and columns that allow for entering numbers or text. Ranges of cells can be specified, modified, and moved. Cell display can be modified, and cell contents can be edited.

This widget does not appear in the Photon documentation.

Instance Variables

matrix_cols

An integer specifying the number of columns in the matrix.

matrix_flags

For internal use only.

matrix_offset

For internal use only. A PhPoint specifying the row/column offset of the upper left matrix cell.

matrix_range

Read only. A PhRect indicating the currently selected range.

matrix_rows

An integer indicating the number of rows in the matrix.

matrix_visible

Read only. An integer indicating the range of visible cells. Current implementation specifies that this will be the whole range.

Callbacks

The following callbacks are associated with this widget. For information on the callback class instance variables, please refer to CwMatrixCallback.

Callback	Description

Callback	Description
Cw_CB_MATRIX_BEGIN_EDIT	This callback is generated when the user has begun to edit the contents of a cell. The program may choose to modify the text_string in order to have the user edit something other than the current cell contents. The relevant callback class instance variables are cur_range, prev_string, text_string and cell.
Cw_CB_MATRIX_CELL_CHANGE	This callback is generated when the user or program has changed something about a cell. The relevant callback class instance variables are cur_range and cell. cur_range simply contains the coordinates of the cell.
Cw_CB_MATRIX_CHARINPUT	This callback is generated when the user has typed a character of input. The relevant callback class instance variables are prev_range, cur_range, prev_string, text_string and cell. The cbinfo.reason_subtype can be Cw_MATRIX_INPUT_BEGIN_EDIT, Cw_MATRIX_INPUT_EDIT, or Cw_MATRIX_INPUT_SELECT.
Cw_CB_MATRIX_KEYPRESS	This callback is generated when the user has pressed a key while not editing a cell. This callback will only be invoked if the key is not a printable character. When the user presses a printable character, editing is automatically started in the currently selected cell. Relevant callback structure elements are cur_range and cell. The key press information can be found in the callback function through the function global variable "event_data" as event_data.key_event.
Cw_CB_MATRIX_RANGE_CONVERT	This callback is generated when the CwMatrix widget requires a printable representation for a range. The range is stored in the callback structure cur_range element, and the suggested printable representation is stored in text_string. The programmer can set text_string to an alternate representation in this callback. Relevant callback class instance variables are cur_range and text_string.
Cw_CB_MATRIX_RANGE_MOVE	This callback is generated when the currently selected range is moved through a dragging operation. The callback is performed only when the user releases the mouse button at the end of the move operation. Relevant callback class instance variables are prev_range and cur_range.

Callback	Description
Cw_CB_MATRIX_RANGE_SEL	This callback is generated when a new range has been selected by the user dragging the mouse. The relevant callback class instance variables are prev_range and cur_range. The cbinfo.reason_subtype can be Cw_MATRIX_RANGE_START, Cw_MATRIX_RANGE_MODIFY or Cw_MATRIX_RANGE_FINAL.
Cw_CB_MATRIX_SIZE_CHANGE	This callback is generated when a row or column is resized. The row and column are stored in cur_range.ul.y and cur_range.ul.x respectively in the callback structure. The width or height of the relevant row or column is stored in the dim element of the callback structure. If a row is resized, the column and width will be reported as -1, and if a column is resized, the row and height will be reported as -1.
Cw_CB_MATRIX_TEXT_CHANGE	This callback is generated when the user has changed the text in a cell. The relevant callback class instance variables are prev_range, cur_range, prev_string, cur_string and cell.

Associated Classes

CwMatrixCallback, CwMatrixCell, PtCallbackInfo

Methods

Arguments

area	A PhArea, which you can create with a call to: new (PhArea).
col	The column number, as an integer.
flags	One or more CwMatrixCell.flags, depending on the method (see below).
flush	An integer. Any non-zero value immediately updates the screen.
height	The height of a cell, in pixels.
inverted	0 or 1.
mask	One or more CwMatrixCell.flags, depending on the method (see below).
range	A PhRect.
row	The row number, as an integer.
width	The width of a cell, in pixels.

Methods

matrix.Cell (*row, col*) -- returns the cell at the position (*row*, *col*). If there is no cell

currently at that position, one is created and returned. If the (*row*,*col*) coordinates are beyond the size of the matrix, returns `nil`.

`matrix.ColumnWidth` (*`col`*) -- returns the width of *`col`*, the specified column, in pixels.

`matrix.DamageCell` (*`row, col`*) -- damages the cell at (*`row`*,*`col`*). The screen is not updated until you call `FlushDamage`.

`matrix.Edit` (*`row, col`*) -- allows for editing the cell at (*`row`*, *`col`*).

`matrix.ExistingCell` (*`row, col`*) -- returns the cell at the position (*`row`*, *`col`*). If there is no cell currently at that position, it returns `nil`. If the (*`row`*, *`col`*) coordinates are beyond the size of the matrix, return `nil`.

`matrix.Flush` () -- is a synonym for matrix.FlushDamage.

`matrix.FlushDamage` () -- causes all damage to be updated on the screen. Many functions delay updating screen damage for efficiency reasons.

`matrix.GetCellArea` (*`row, col, area`*) -- fills in the elements of the *`area`*. This gives the pixel position and area of the cell at (*`row`*,*`col`*). The *`area`* is relative to the CwMatrix widget. The method returns -1 if the cell is completely off-screen, otherwise 0.

`matrix.GetCellExtent` (*`row, col, area`*) -- fills in the elements of the provided PhArea. This gives the pixel position and area of the cell at (*`row`*,*`col`*). The area is relative to the parent of the CwMatrix widget. The method returns -1 if the cell is completely off-screen, otherwise 0.

`matrix.InvertRange` (*`range, inverted`*) -- inverts or un-inverts the given range.

`matrix.JustifyRange` (*`range, flags`*) -- sets the text justification on all cells in the range. The *`flags`* parameter can be one of: Cw_CELL_LEFT, Cw_CELL_RIGHT or Cw_CELL_CENTER.

`matrix.MaskRange` (*`range, flags, mask`*) -- generally sets or un-sets flags on a range. The *`flags`* and *`mask`* parameters can be any combination of Cw_CELL_* flags.

`matrix.OutlineRange` (*`range, flags, mask`*) -- draws a border around the selected range. This is not the same as MaskRange, as it draws the border only on the outside edge of the entire range, whereas MaskRange will apply the flags and mask to every cell in the range. The *`flags`* and *`mask`* parameters can be any combination of Cw_CELL_* flags.

`matrix.Redraw` () -- redraws the entire widget.

`matrix.RowHeight` (*`row`*) -- returns the height of the specified *`row`*, in pixels.

`matrix.SetColumnWidth` (*`col, width, flush`*) -- sets the *`width`* in pixels for *`col`*, the specified column. If *`flush`* is non-zero, it causes the screen to be updated immediately.

`matrix.SetRowHeight` (*`row, height, flush`*) -- sets the *`height`* in pixels for the specified *`row`*. If *`flush`* is non-zero, it causes the screen to be updated immediately.

Example

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

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

/*
 * This example illustrates CwMatrix and CwMatrixCell.
 */

/* Load the required files, which include two dynamic
 * libraries that are not generally needed for Photon
 * widgets.
 */
if (_os_ == "QNX4")
```

```
{
  dyna_add_lib("/usr/cogent/lib/photon_s.dlb");
  dyna_add_lib("/usr/cogent/lib/phwidgets.dlb");
}
require_lisp("PhotonWidgets.lsp");
PtInit(nil);

/* Create the Matrix.
 */
win = new(PtWindow);
win.SetArea(340,10,200,100);
mtx = new(CwMatrix);
mtx.SetArea(25,20,150,70);
mtx.matrix_cols = 3;
mtx.matrix_rows = 3;

/* Demonstrate some methods.
 */
w = mtx.ColumnWidth(0);
h = mtx.RowHeight(2);
cellarea = new(PhArea);
mtx.GetCellArea(2,2,cellarea);
princ("The width of the first column is ",w,".\n");
princ("The height of the bottom row is ",h,".\n\n");
pretty_princ("The bottom-right cell area is: \n",
      cellarea,"\n\n");

/* Specify a range, and invert its colors.
 */
ul = new(PhPoint);
lr = new(PhPoint);
ul.x = 0;
ul.y = 0;
lr.x = 2;
lr.y = 0;
range = new(PhRect);
range.ul = ul;
range.lr = lr;
pretty_princ("The inverted range is:\n",range,"\n\n");
mtx.InvertRange(range,1);

/* Demonstrate a CwMatrixCell.
 */
cell = mtx.Cell(1,1);
cell.text_string = "Test";
cell.text_color = 0x0000ff;
cell.fill_color = 0xffffaa;
pretty_princ("The center cell definition is: \n",cell,"\n\n");

/* Demonstrate some callbacks.
 */
PtAttachCallback(mtx,Cw_CB_MATRIX_RANGE_MOVE,
                 #princ("Range has moved.\n"));
PtAttachCallback(mtx,Cw_CB_MATRIX_TEXT_CHANGE,
                 #princ("Text has changed.\n"));

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

Also see A CwMatrix Spreadsheet in the Sample Code and Cool Stuff chapter of the Programmer's Manual for a more sophisticated example.

PtArc

PtArc — An elliptical arc.

Synopsis

```
class PtArc PtGraphic
{
    arc_end;      // unsigned short  (Pt_ARG_ARC_END)
    arc_start;    // unsigned short  (Pt_ARG_ARC_START)
    arc_type;     // unsigned short  (Pt_ARG_ARC_TYPE)
}
```

Base Classes

PtWidget <-- PtBasic <-- PtGraphic <-- PtArc

Description

This widget is an arc of an ellipse, defined by its point of origin, bounding rectangle, and start and end angles.

The point of origin accesses the inherited origin variable, which is a single PtPoint. The bounding rectangle is determined by two PtPoints (relative to the origin), or by the values of the inherited dim variable. The default arc is a full circle with start and end points at the same point: 0 degrees, the point horizontal to the centroid of the ellipse.

There are three types of fill: empty (open curve), chord, or pie-shaped, specified by their respective variables as explained below.

For detailed information, please refer to PtArc in the Photon documentation.

Instance Variables

arc_end

A number specifying the end angle of the arc, in tenths of degrees.

arc_start

A number specifying the start angle of the arc, in tenths of degrees.

arc_type

A constant that controls the type of arc: open curve, closed pie-shaped, or closed with a chord. Default is Pt_ARC_CURVE.

This instance variable may have one of the following values:

Constant	Description
Pt_ARC_CURVE	An unclosed, unfilled curve; the default.
Pt_ARC_PIE	A curve closed by two lines that connect the end points to the centroid of the arc, like a piece of pie.
Pt_ARC_CHORD	A curve closed by a line connecting the two end points.

PtBarGraph

PtBarGraph — A bar graph.

Synopsis

```
class PtBarGraph PtBasic
{
    bargraph_base;          // short
    bargraph_color;         // color array
    bargraph_data;          // bar data array
    bargraph_depth;         // short
    bargraph_flags;         // flag
    bargraph_grid_color;    // color
    bargraph_grid_horiz;    // short
    bargraph_grid_vert;     // short
    bargraph_max;           // short
    bargraph_min;           // short
}
```

Base Classes

PtWidget <-- PtBasic <-- PtBarGraph

Description

This widget creates a bar graph from a data array. It features an adjustable baseline, user-assigned bar colors, and an optional grid. The bars automatically adjust their width to fill the width of the widget.

This widget does not appear in the Photon documentation.

Instance Variables

`bargraph_base`

A number specifying a baseline for the graph display. Any `bargraph_data` value lower than the baseline value will display a bar going down from the baseline, while any `bargraph_data` value greater than the baseline will display a bar going up from the baseline. The default baseline is 0.

`bargraph_color`

An array of numbers specifying colors for each value in `bargraph_data`. The default color is red, but if at least one color is assigned, Gamma chooses among several shades of blue, grey and black for the remaining colors.

`bargraph_data`

An array of numbers corresponding to the data to be graphed.

`bargraph_depth`

An integer specifying the depth of the highlighted border to give to each bar. Default is `0`.

`bargraph_flags`

Flags that allow you to specify various options, of which only one is currently available.

This instance variable may be a combination of zero or more of the following flags:

Constant	Description
`Pt_BARGRAPH_GRID`	Puts a grid on the bargraph. Default is ON.

`bargraph_grid_color`

An integer specifying the color of the grid lines, displayed if the `Pt_BARGRAPH_GRID` flag is set on the `bargraph_flags` variable. Default is `0x606060` (dark grey).

`bargraph_grid_horiz`

An integer specifying the number of horizontal grid lines if a grid is to be displayed. Default is `6`.

`bargraph_grid_vert`

An integer specifying the number of vertical grid lines if a grid is to be displayed. Default is `6`.

`bargraph_max`

An integer specifying the maximum value for the graph. Bars for numbers above this value will be truncated at the top of the graph. The highest allowable value is 32,767, which is also the default.

`bargraph_min`

An integer specifying the minimum value for the graph. Bars for numbers below this range will be truncated at the bottom of the graph. The lowest allowable value is `-32,768`, which is also the default.

Example

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

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

/*
 * This example demonstrates a PtBarGraph, using data with steadily
 * decreasing values. The baseline is set to -50, and green_tinted colors
 * have been specified for all the positive numbers.  The 39 horizontal
 * grid lines divide the graph into 40 equal horizontal sections, while
 * the 8 vertical gridlines make 9 equal vertical sections.
 */
require_lisp("PhotonWidgets.lsp");
PtInit(nil);

win = new(PtWindow);
win.SetDim (325,325);

bgf = new(PtBarGraph);
bgf.SetArea (10,10,300,300);
bgf.fill_color = 0xaabfaa;
bgf.bargraph_base = -50;
bgf.bargraph_color = array(0xcceecc, 0xaaeeaa, 0x77ee77, 0x55ee55);
bgf.bargraph_data = array(150,100,50,25,0,-25,-50,-100,-150);
bgf.bargraph_depth = 3;
bgf.bargraph_flags = Pt_BARGRAPH_GRID;
bgf.bargraph_grid_color = 0x00aaff;
bgf.bargraph_grid_horiz = 8;
bgf.bargraph_grid_vert = 39;
bgf.bargraph_max = 200;
bgf.bargraph_min = -200;

princ(hex(bgf.bargraph_max),"\n");
princ(bgf.bargraph_min,"\n");

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

PtBasic

PtBasic — A parent class for basic widget resources.

Synopsis

```
class PtBasic PtWidget
{
    basic_flags;            // flag
    bot_border_color;       // color  (Pt_ARG_BOT_BORDER_COLOR)
    color;                  // color  (Pt_ARG_COLOR)
    fill_color;             // color  (Pt_ARG_FILL_COLOR)
    fill_pattern;           // pattern string  (Pt_ARG_FILL_PATTERN)
    highlight_roundness;    // unsigned short  (Pt_ARG_HIGHLIGHT_ROUNDNESS)
    margin_height;          // unsigned short  (Pt_ARG_MARGIN_HEIGHT)
    margin_width;           // unsigned short  (Pt_ARG_MARGIN_WIDTH)
    top_border_color;       // color  (Pt_ARG_TOP_BORDER_COLOR)
    trans_pattern;          // pattern string  (Pt_ARG_TRANS_PATTERN)
}
```

Base Classes

PtWidget <-- PtBasic

Derived Classes

CwGraph, PtBarGraph, PtBitmap, PtCalendar, PtClock, PtContainer, PtFolderTab, PtGauge, PtGraphic, PtGrid, PtLabel, PtLed, PtRaw, PtScrollbar, PtSeparator, RtMeter, RtTrend

Description

This class serves as a parent class of resources, and is not normally instantiated. The variables here have to do mainly with physical appearance such as color, patterns, and margin sizes.

For detailed information, please refer to PtBasic in the Photon documentation.

Instance Variables

`basic_flags`

Not yet implemented.

`bot_border_color`

An integer specifying the color of the bottom and right borders. Default is `0x0` (black).

`color`

An integer specifying the color used for drawing (foreground).

`fill_color`

An integer specifying the color used as a background. Default is `0xc0c0c0` (grey).

`fill_pattern`

A string specifying the fill pattern. Default is `Pg_PAT_FULL`.

`highlight_roundness`

A number specifying the radius in pixels of the corners of a rectangular widget. Default is `0`.

`margin_height`

The number of pixels between the drawing area and the top and bottom of the widget. Default is `2`.

`margin_width`

The number of pixels between the drawing area and the left and right sides of the widget. Default is `2`.

`top_border_color`

An integer specifying the color of the top and left borders. Default is `0xffffff` (white).

`trans_pattern`

A string specifying the transparency pattern, useful for ghosting images.

Callbacks

The following callbacks are associated with this widget:

Callback	Description
`Pt_CB_ARM`	This callback is generated when the widget is armed.
`Pt_CB_DISARM`	This callback is generated when the widget is disarmed.
`Pt_CB_ACTIVATE`	This callback is generated when the widget is activated.
`Pt_CB_GOT_FOCUS`	This callback is generated when the widget gets focus or changes focus status.
`Pt_CB_LOST_FOCUS`	This callback is generated when the widget loses focus.
`Pt_CB_REPEAT`	This callback is generated when the widget receives but_repeat events.
`Pt_CB_MENU`	This callback is generated when the pointer is over the widget and the right button is pressed.

Associated Classes

`PtBasicCallback`, `PtCallbackInfo`

Convenience Functions

Arguments

`widget`	A `PtBasic` widget.

Functions

This function is an extension of the QNX Photon function. You can refer to the PtBasic function documentation in QNX Helpviewer for more information about it.

`PtBasicWidgetCanvas` `(widget)` -- determines the area inside the *`widget`*'s border.

Returns a `PtRect`.

PtBezier

PtBezier — A bezier curve.

Synopsis

```
class PtBezier PtGraphic
{
    bezier_flags;    // flag  (Pt_ARG_BEZIER_FLAGS)
}
```

Base Classes

PtWidget <-- PtBasic <-- PtGraphic <-- PtBezier

Description

This widget creates a bezier curve, most of whose characteristics can be specified with variables from the PtWidget, PtBasic, and PtGraphic classes.

For detailed information, please refer to PtBezier in the Photon documentation.

Instance Variables

bezier_flags

Flags that control various characteristics of the curve. Default is nil.

This instance variable may be a combination of zero or more of the following flags:

Constant	Description
Pg_DRAW_STROKE	Strokes the curve.
Pg_DRAW_FILL	Fills the curve.
Pg_DRAW_FILL_STROKE	Fills and strokes the curve.
Pg_CLOSED	The end point is connected to the start point.
Pg_RELATIVE	Use relative coordinates for drawing.

PtBitmap

PtBitmap — A bitmapped image.

Synopsis

```
class PtBitmap PtBasic
{
    bitmap_balloon;              // PtWidget  (Pt_ARG_BITMAP_BALLOON)
    bitmap_balloon_color;        // color  (Pt_ARG_BITMAP_BALLOON_COLOR)
    bitmap_balloon_fill_color;   // color  (Pt_ARG_BITMAP_BALLOON_FILL_COLOR)
    bitmap_balloon_position;     // short  (Pt_ARG_BITMAP_BALLOON_POSITION)
    bitmap_colors;               // color array  (Pt_ARG_BITMAP_COLORS)
    bitmap_data;                 // char array array  (Pt_ARG_BITMAP_DATA)
    bitmap_flags;                // flag  (Pt_ARG_BITMAP_FLAGS)
    bitmap_text;                 // string  (Pt_ARG_BITMAP_TEXT)
    set_bg_color;                // color  (Pt_ARG_BMP_SET_BG_COLOR)
    set_bg_fill;                 // unsigned char  (Pt_ARG_BMP_SET_BG_FILL)
    set_bitmap_colors;           // color array  (Pt_ARG_SET_BITMAP_COLORS)
    set_bitmap_data;             // char array array  (Pt_ARG_SET_BITMAP_DATA)
}
```

Base Classes

PtWidget <-- PtBasic <-- PtBitmap

Description

This widget is used to draw bitmaps, but is difficult to create and use in code. The easiest way to work with images in Gamma is with the PhImage class, which lets you create images in other applications and attach them to a PtPane or PtButton as necessary.

If for some reason you need a PtBitmap, it is fully supported in Gamma, but we recommend creating and modifying it in PhAB, not in code.

For detailed information, please refer to PtBitmap in the Photon documentation.

Instance Variables

bitmap_balloon

bitmap_balloon_color

bitmap_balloon_fill_color

bitmap_balloon_position

This instance variable may have one of the following values:

Constant	Description
Pt_BITMAP_BALLOON_RIGHT	The text will pop up on the right.
Pt_BITMAP_BALLOON_LEFT	The text will pop up on the left.
Pt_BITMAP_BALLOON_TOP	The text will pop up at the top.
Pt_BITMAP_BALLOON_BOTTOM	The text will pop up at the bottom.
Pt_BITMAP_BALLOON_INPLACE	The text will pop up in place.

`bitmap_colors`

`bitmap_data`

`bitmap_flags`

This instance variable may be a combination of zero or more of the following flags:

Constant	Description
`Pt_BITMAP_SHOW_BALLOON`	Causes balloon help to be shown if the cursor remains stationary over the bitmap for 1.25 seconds.

`bitmap_text`

`set_bg_color`

`set_bg_fill`

`set_bitmap_colors`

`set_bitmap_data`

PtBkgd

PtBkgd — A background image, bitmap, or color-gradient.

Synopsis

```
class PtBkgd PtPane
{
    bkgd_brt_from;       // unsigned short  (Pt_ARG_BKGD_BRT_FROM)
    bkgd_brt_to;         // unsigned short  (Pt_ARG_BKGD_BRT_TO)
    bkgd_hue_from;       // unsigned short  (Pt_ARG_BKGD_HUE_FROM)
    bkgd_hue_to;         // unsigned short  (Pt_ARG_BKGD_HUE_TO)
    bkgd_image;          // PhImage  (Pt_ARG_BKGD_IMAGE)
    bkgd_mix;            // unsigned short  (Pt_ARG_BKGD_MIX)
    bkgd_orientation;    // unsigned short  (Pt_ARG_BKGD_ORIENTATION)
    bkgd_pix_height;     // unsigned short  (Pt_ARG_BKGD_PIX_HEIGHT)
    bkgd_pix_width;      // unsigned short  (Pt_ARG_BKGD_PIX_WIDTH)
    bkgd_pixcolors;      // color array  (Pt_ARG_BKGD_PIXCOLORS)
    bkgd_pixmap;         // char array array  (Pt_ARG_BKGD_PIXMAP)
    bkgd_sat_from;       // unsigned short  (Pt_ARG_BKGD_SAT_FROM)
    bkgd_sat_to;         // unsigned short  (Pt_ARG_BKGD_SAT_TO)
    bkgd_spacing;        // PhPoint  (Pt_ARG_BKGD_SPACING)
    bkgd_steps;          // unsigned short  (Pt_ARG_BKGD_STEPS)
    bkgd_tile;           // unsigned short  (Pt_ARG_BKGD_TILE)
    bkgd_type;           // unsigned short  (Pt_ARG_BKGD_TYPE)
}
```

Base Classes

PtWidget <-- PtBasic <-- PtContainer <-- PtPane <-- PtBkgd

Description

This widget lets you make backgrounds for application windows. The bkgd_type variable lets you choose from three types of backgrounds: images, pixmaps, and color-gradients, which are controlled with their associated variables.

- **Images** can be created in graphics programs, loaded with a call to PxLoadImage, and subsequently used for background images as instances of PhImage. They can be tiled in several ways using the bkgd_tile and bkgd_spacing variables.
- **Pixmaps** are bitmapped images created in PhAB, or by using the Photon function PtDrawBitmap. This type of image can easily be made transparent.
- **Color gradients** are specified by their hue, saturation, and brightness variables: bkgd_hue, bkgd_sat and bkgd_brt, each with a _from and _to variable for specifying the range. But the range can only be applied to one of the three. Choose one of them, and set the flag for it in the bkgd_type variable. Then for that one alone you can specify the range, assigning both its _from and _to variables. The other two take only the _to variable.

  Other options for color gradients include horizontal or vertical orientation, color dithering, and number of gradient steps.

For detailed information, please refer to PtBkgd in the Photon documentation.

Instance Variables

`bkgd_brt_from`, bkgd_brt_to

An integer between 0 and 255 specifying the brightness range (black to full brightness) of color gradients, using the hue/saturation/brightness color model.

`bkgd_hue_from`, bkgd_hue_to

An integer between 0 and 65535 specifying the hue range (on the color spectrum) of color gradients, using the hue/saturation/brightness color model.

`bkgd_image`

A `PhImage` that you wish to display.

`bkgd_mix`

An integer that specifies whether or not dithering is used in color-gradient backgrounds. 1 (the default) is on, 0 is off.

`bkgd_orientation`

A constant specifying the orientation of color-gradient backgrounds. The default is vertical.

This instance variable may have one of the following values:

Constant	Description
`Pt_BKGD_HORIZONTAL`	Sets the gradient orientation to horizontal.
`Pt_BKGD_VERTICAL`	Sets the gradient orientation to vertical.

`bkgd_pix_height`, bkgd_pix_width

A number of pixels specifying the height and width of a pixmap image.

`bkgd_pixcolors`

An array of colors used by a pixmap image.

`bkgd_pixmap`

A bitmapped background image created in PhAB. This type of image can be easily made transparent.

`bkgd_sat_from`, bkgd_sat_to

An integer between 0 and 255 specifying the saturation range (white to full color) of color gradients, using the hue/saturation/brightness color model.

`bkgd_spacing`

A `PhPoint` whose x and y values specify the horizontal and vertical separation between tiles. Default is:

```
{PhPoint (x . 0) (y . 0)}
```

`bkgd_steps`

An integer specifying the number of steps to get from the minimum gradient value to the maximum. Default is `10`.

`bkgd_tile`

A constant specifying the tiling option for images and pixmaps.

This instance variable may have one of the following values:

Constant	Description
Pt_BKGD_NONE	None.
Pt_BKGD_GRID	The image repeats in rows and columns.
Pt_BKGD_ALT	The image repeats alternately, like brickwork.
Pt_BKGD_CENTER	The image is drawn in the center of the container.
Pt_BKGD_CENTER_GRID	The image is drawn in the center, and repeated for a single row and column around the border.

bkgd_type

A constant specifying the type of background image you want to use. The relevance of other variables depends on this choice.

This instance variable may have one of the following values:

Constant	Description
Pt_BKGD_HUE	The background color gradient is based on hue value.
Pt_BKGD_SATURATION	The background color gradient is based on saturation value.
Pt_BKGD_BRIGHTNESS	The default, the background color gradient is based on brightness value.
Pt_BKGD_PIXMAP	Used to create a background of a small/hand-drawn image; background fill color stays visible.
Pt_BKGD_IMAGE	Used to create a background with an imported image.

Example

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

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

/*
 * This example demonstrates a saturation color_gradient
 * PtBkgd anchored to a window.
 */

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

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

back = new(PtBkgd);
back.bkgd_type = Pt_BKGD_SATURATION;
back.bkgd_brt_to = 255;
back.bkgd_hue_to = 30000;
back.bkgd_sat_from = 255;
```

```
back.bkgd_sat_to = 0;
back.bkgd_steps = 200;
back.SetDim(300,300);
back.bkgd_orientation = Pt_BKGD_HORIZONTAL;
back.anchor_flags = Pt_LEFT_ANCHORED_RELATIVE |
                    Pt_RIGHT_ANCHORED_RELATIVE |
      Pt_TOP_ANCHORED_RELATIVE |
      Pt_BOTTOM_ANCHORED_RELATIVE;

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

PtButton

PtButton — A push-button used to initiate a callback.

Synopsis

```
class PtButton PtLabel
{
    arm_color;    // color  (Pt_ARG_ARM_COLOR)
    arm_fill;     // unsigned char  (Pt_ARG_ARM_FILL)
    arm_image;    // PhImage  (Pt_ARG_ARM_DATA)
}
```

Base Classes

PtWidget <-- PtBasic <-- PtLabel <-- PtButton

Description

This widget is a button that can be used to initiate actions in Gamma, usually through callbacks. When a button is pressed (the mouse button is held down on it), it is *armed*, and its color becomes shaded. As a child of the PtLabel class, a PtButton inherits the label_type variable, which allows you to put text strings, images, and bitmaps on it. What's more, you can have a PtButton change its color or image when it is armed, by setting its own instance variables.

For detailed information, please refer to PtButton in the Photon documentation.

Instance Variables

arm_color

An integer specifying the color to change the button to when it is pressed. Default is 0xaaaaaa (dark grey).

arm_fill

An integer specifying whether or not the arm_color will be used. 1 (the default) means yes, 0 means no.

arm_image

A PhImage to display when the button is pressed.

PtCalendar

PtCalendar — A month-based calendar.

Synopsis

```
class PtCalendar PtBasic
{
    calendar_color1;            // color  (Pt_ARG_CALENDAR_COLOR1)
    calendar_color2;            // color  (Pt_ARG_CALENDAR_COLOR2)
    calendar_color3;            // color  (Pt_ARG_CALENDAR_COLOR3)
    calendar_color4;            // color  (Pt_ARG_CALENDAR_COLOR4)
    calendar_color5;            // color  (Pt_ARG_CALENDAR_COLOR5)
    calendar_date;              // long  (Pt_ARG_CALENDAR_DATE)
    calendar_flags;             // flag  (Pt_ARG_CALENDAR_FLAGS)
    calendar_font1;             // string  (Pt_ARG_CALENDAR_FONT1)
    calendar_font2;             // string  (Pt_ARG_CALENDAR_FONT2)
    calendar_font3;             // string  (Pt_ARG_CALENDAR_FONT3)
    calendar_font4;             // string  (Pt_ARG_CALENDAR_FONT4)
    calendar_font5;             // string  (Pt_ARG_CALENDAR_FONT5)
    calendar_highlight;         // long  (Pt_ARG_CALENDAR_HIGHLIGHT)
    calendar_month_btn_color;   // color  (Pt_ARG_CALENDAR_MONTH_BTN_COLOR)
    calendar_month_names;       // string array  (Pt_ARG_CALENDAR_MONTH_NAMES)
    calendar_sel_color;         // color  (Pt_ARG_CALENDAR_SEL_COLOR)
    calendar_time_t;            // long  (Pt_ARG_CALENDAR_TIME_T)
    calendar_wday_names;        // string array  (Pt_ARG_CALENDAR_WDAY_NAMES)
    calendar_year_btn_color;    // color  (Pt_ARG_CALENDAR_YEAR_BTN_COLOR)
}
```

Base Classes

PtWidget <-- PtBasic <-- PtCalendar

Description

This widget is a 6-line calendar that displays the days of the week for one month, and the year. You can control various aspects of the display, such as date colors, button colors, fonts, and name displays.

For detailed information, please refer to PtCalendar in the Photon documentation.

Instance Variables

`calendar_color1`

A number specifying the display color for the numerals of the days of the current month. Default is `0x000000` (black).

`calendar_color2`

A number specifying the display color for the numerals of the days of the next and previous months. Default is `0x606060` (dark grey).

`calendar_color3`

A number specifying the display color for the year and month name characters. Default is `0x000000` (black).

`calendar_color4`

A number specifying the display color for highlighted days. Default is `0x000000` (black).

`calendar_color5`

A number specifying the display color of the characters in the names of the days of the week. Default is `0x0000ff` (blue).

`calendar_date`

A `PtCalendarDate` date designated as the current date.

`calendar_flags`

Flags specifying display options, such as buttons for accessing previous and next years, months, and days, and a grid that separates the days and weeks.

This instance variable may be a combination of zero or more of the following flags:

Constant	Description
`Pt_CALENDAR_YEAR_BTNS`	Show the buttons for the next and previous years. Default is ON.
`Pt_CALENDAR_MONTH_BTNS`	Show the buttons for the next and previous months. Default is ON.
`Pt_CALENDAR_SHOW_PREV`	Show the final days of the previous month. Default is ON.
`Pt_CALENDAR_SHOW_NEXT`	Show the first days of the next month. Default is ON.
`Pt_CALENDAR_SHOW_GRID`	Show the calendar as a grid, one block per day. Default is ON.

`calendar_font1`

A string specifying the font to use for the days of the current month. The default is "lu12".

`calendar_font2`

A string specifying the font to use for the days of the next and previous month. The default is "lu12i".

`calendar_font3`

A string specifying the font to use for the year, and for the month names. The default is "lu12b".

`calendar_font4`

A string specifying the font to use for highlighted days. The default is "lu12b".

`calendar_font5`

A string specifying the font to use for the names of the days of the week. The default is "helv12b".

`calendar_highlight`

An integer specifying highlighted days. Default is `0`.

`calendar_month_btn_color`

An integer specifying the color of the previous/next month buttons. Default is `0xc0c0c0` (grey).

`calendar_month_names`

A write-only array of exactly 12 elements (strings) that are the names of the months. Assigning a new array will replace the existing values. If the new array has fewer than 12 elements, missing elements will be supplied by the existing array. If the new array has more than 12 elements, the extra elements will not be used.

`calendar_sel_color`

An integer specifying the color of the selected date. Default is `0xffff00` (yellow).

`calendar_time_t`

A number specifying the current date on the calendar.

`calendar_wday_names`

A write-only array of exactly 7 elements:

[Su Mo Tu We Th Fr Sa]

that abbreviate the names of the days of the week. Assigning a new array will replace the existing values. If the new array has fewer than 7 elements, the missing elements will be supplied by the existing array. If the new array has more than 7 elements, the extra elements will not be used.

`calendar_year_btn_color`

An integer specifying the color of the previous/next year buttons. Default is `0xc0c0c0` (grey).

Callbacks

The following callback is associated with this widget:

Callback	Description
`Pt_CB_CALENDAR_SELECT`	This callback is generated when a date is selected.

Associated Classes

`PtCalendarDate`, `PtCalendarSelectCallback`, `PtCallbackInfo`

Example

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

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

/*
 * This example demonstrates a customized PtCalendar.
 */

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

win = new(PtWindow);
cal = new(PtCalendar);

cal.SetDim(350,200);
cal.fill_color = 0xffddcc;
cal.calendar_font1 = "lu12b";
cal.calendar_color3 = 0x0000ff;
cal.calendar_color5 = 0xbb00bb;
cal.calendar_flags = cons(Pt_CALENDAR_SHOW_GRID,nil);
cal.calendar_month_btn_color = 0xff0000;
cal.calendar_sel_color = 0xffff00;
cal.calendar_wday_names = array("Sun","Mon","Tues","Wed","Thurs","Fri","Sat");

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

PtClock

PtClock — A clock with analog, digital or LED-style format.

Synopsis

```
class PtClock PtBasic
{
    clock_face_color;           // color  (Pt_ARG_CLOCK_FACE_COLOR)
    clock_face_outline_color;   // color  (Pt_ARG_CLOCK_FACE_OUTLINE_COLOR)
    clock_flags;                // flag  (Pt_ARG_CLOCK_FLAGS)
    clock_font;                 // string  (Pt_ARG_CLOCK_FONT)
    clock_hour;                 // short  (Pt_ARG_CLOCK_HOUR)
    clock_hour_color;           // color  (Pt_ARG_CLOCK_HOUR_COLOR)
    clock_hour_offset;          // short  (Pt_ARG_CLOCK_HOUR_OFFSET)
    clock_minute;               // short  (Pt_ARG_CLOCK_MINUTE)
    clock_minute_color;         // color  (Pt_ARG_CLOCK_MINUTE_COLOR)
    clock_minute_offset;        // short  (Pt_ARG_CLOCK_MINUTE_OFFSET)
    clock_second;               // short  (Pt_ARG_CLOCK_SECOND)
    clock_second_color;         // color  (Pt_ARG_CLOCK_SECOND_COLOR)
    clock_second_offset;        // short  (Pt_ARG_CLOCK_SECOND_OFFSET)
    clock_sep1;                 // string  (Pt_ARG_CLOCK_SEP1)
    clock_sep1_color;           // color  (Pt_ARG_CLOCK_SEP1_COLOR)
    clock_sep2;                 // string  (Pt_ARG_CLOCK_SEP2)
    clock_sep2_color;           // color  (Pt_ARG_CLOCK_SEP2_COLOR)
    clock_type;                 // short  (Pt_ARG_CLOCK_TYPE)
}
```

Base Classes

PtWidget <-- PtBasic <-- PtClock

Description

This widget displays a clock in one of three types: analog (with a dial and hands), digital, or LED format. The display is good for general time-keeping but is not very precise. It is updated about every second, and is prone to flickering. You can minimize this annoyance by not using a transparent fill color, by disabling the seconds, and/or by putting the clock in a PtContainer widget.

For detailed information, please refer to PtClock in the Photon documentation.

Instance Variables

clock_face_color

An integer specifying the color of an analog clock face. Default is 0xffffff (white).

clock_face_outline_color

An integer specifying the color of the line around an analog clock face. Default is 0x0 (black).

clock_flags

Flags that define the clock's appearance and behavior. The default is PtCLOCK_TRACK_TIME | Pt_CLOCK_SHOW_SECONDS | Pt_CLOCK_SHOW_NUMBERS.

This instance variable may be a combination of zero or more of the following flags:

Constant	Description
Pt_CLOCK_TRACK_TIME	Updates the clock to current time, by the second.

Constant	Description
Pt_CLOCK_SHOW_SECONDS	Shows the seconds of the time.
Pt_CLOCK_24_HOUR	Makes a 24-hour display, instead of 12-hour.
Pt_CLOCK_SHOW_NUMBERS	Sets an analog clock to display numbers instead of hands.
Pt_CLOCK_SHOW_AMPM	Adds an AM/PM display to digital and LED clocks.
Pt_CLOCK_PAD_HOURS	Adds a leading zero to one-digit hour displays in digital and LED clocks.

clock_font

A string specifying the font for numbers on analog clocks, and for the whole display for digital clocks.

clock_hour, clock_minute, clock_second

An integer indicating the current hour, minute, or second setting of the clock, in 24-hour format. The default for each of these is Pt_CLOCK_CURRENT, which sets to system time when the widget is realized.

clock_hour_color, clock_minute_color, clock_second_color

An integer specifying the color for the hour, minute, or second display. The defaults are 0x0 (black), 0x0 , and 0xff0000 (red) respectively.

clock_hour_offset, clock_minute_offset, clock_second_offset

An integer specifying the number of hours, minutes, or seconds to offset the respective field in the display, if desired.

clock_sep1, clock_sep1_color

A string and a number specifying the character and color of the separator between the hours and the minutes on a digital clock. The defaults are 0x0 (black) and ":".

clock_sep2, clock_sep2_color

A string and a number specifying the character and color of the separator between the minutes and the seconds on a digital clock. The defaults are 0x0 (black) and ":".

clock_type

This instance variable specifies the type of clock, and may have one of the following values:

Constant	Description
Pt_CLOCK_DIGITAL	The clock has a digital display that uses system fonts.
Pt_CLOCK_ANALOG	The clock has an analog display: a dial with hands.
Pt_CLOCK_LED	The clock has a display similar to an LED or LCD clock, which can be scaled.

Callbacks

The following callback is associated with this widget:

Callback	Description
Pt_CB_CLOCK_TIME_CHANGED	This callback is generated when the time on the clock changes.

Associated Classes

PtClockTimeCallback, PtCallbackInfo

PtComboBox

PtComboBox — A text field and a list of choices.

Synopsis

```
class PtComboBox PtContainer
{
    cbox_button_border_width;        // short  (Pt_ARG_CBOX_BUTTON_BORDER_WIDTH)
    cbox_button_bot_border_color;    // color  (Pt_ARG_CBOX_BUTTON_BOT_BORDER_COLOR)
    cbox_button_color;               // color  (Pt_ARG_CBOX_BUTTON_COLOR)
    cbox_button_top_border_color;    // color  (Pt_ARG_CBOX_BUTTON_TOP_BORDER_COLOR)
    cbox_button_width;               // unsigned short  (Pt_ARG_CBOX_BUTTON_WIDTH)
    cbox_flags;                      // flag  (Pt_ARG_CBOX_FLAGS)
    cbox_max_visible_count;          // unsigned short  (Pt_ARG_CBOX_MAX_VISIBLE_COUNT)
    cbox_sel_item;                   // unsigned short  (Pt_ARG_CBOX_SEL_ITEM)
}
```

Base Classes

PtWidget <-- PtBasic <-- PtContainer <-- PtComboBox

Description

This widget combines a PtText widget with a PtList widget, and allows you to either enter a choice, or choose it from the list. The list can be either static or dropping, and you choose items by clicking on them with the mouse, or dragging down and releasing the mouse button. The list is scrollable if the number of items exceeds a specifiable maximum.

The first five instance variables for this widget are self-explanatory. The button they refer to is the drop button that activates the drop list.

For detailed information, please refer to PtComboBox in the Photon documentation.

Instance Variables

`cbox_button_border_width`

A number of pixels specifying the border width. Default is `2`.

`cbox_button_bot_border_color`

A number specifying the color of the drop button's bottom border. Default is `0xffffff` (white).

`cbox_button_color`

A number specifying the color of the drop button. Default is `0xc0c0c0` (grey).

`cbox_button_top_border_color`

A number specifying the color of the drop button's top border. Default is `0x606060` (dark grey).

`cbox_button_width`

A number of pixels specifying the width of the button. Default is `17`.

`cbox_flags`

Flags that control the behavior of the widget.

This instance variable may be a combination of zero or more of the following flags:

Constant	Description

Constant	Description
Pt_COMBOBOX_STATIC	Changes the field from drop (the default) to static, and removes the drop button.
Pt_COMBOBOX_TOP	Changes the default location of a drop field from the bottom to the top of the text field.
Pt_COMBOBOX_MAX_WIDTH	Maximizes the width of the box to the size of the longest item.
Pt_COMBOBOX_OPEN	When set (1), the list is open.
Pt_COMBOBOX_ON_BOTTOM	Internal informational bit.
Pt_COMBOBOX_EXTENTING	Internal informational bit.

cbox_max_visible_count

An integer specifying the maximum number of list items to display. If the number exceeds this maximum, a scrollbar will appear. If it is set to 0 (the default), all items will be displayed.

cbox_sel_item

An index into the items variable that indicates the selected list item.

Callbacks

The following callbacks are associated with this widget:

Callback	Description
Pt_CB_CBOX_ACTIVATE	This callback is generated when the combo box list is opened.
Pt_CB_CBOX_CLOSE	This callback is generated when the combo box list is closed.

Since this class combines PtList and PtText, it inherits callbacks from both of those classes.

Associated Classes

PtList, PtText

PtContainer

PtContainer — A parent class for container widget resources.

Synopsis

```
class PtContainer PtBasic
{
    anchor_flags;       // flag  (Pt_ARG_ANCHOR_FLAGS)
    anchor_offsets;     // PhRect  (Pt_ARG_ANCHOR_OFFSETS)
    container_flags;    // flag  (Pt_ARG_CONTAINER_FLAGS)
    focus;              // PtWidget  (Pt_ARG_FOCUS)
}
```

Base Classes

PtWidget <-- PtBasic <-- PtContainer

Derived Classes

CwMatrix, PtComboBox, PtDBContainer, PtDivider, PtFolder, PtFontSel, PtGenList, PtGroup, PtHtml, PtMenuButton, PtMessage, PtMultiText, PtNumeric, PtPane, PtRegion, PtScrollArea, PtTerminal, PtUpDown, PtWindow

Description

This widget serves as a parent class of resources for container widgets, and is not normally instantiated. The variables here have to do mainly with geometry, anchoring, layout and redirecting of certain events to child widgets.

For detailed information, please refer to PtContainer in the Photon documentation.

Instance Variables

anchor_flags

This instance variable controls anchoring characteristics, and may be a combination of zero or more of the following flags:

Constant	Description
Pt_LEFT_ANCHORED_RELATIVE	Anchor the left edge proportionally within the width of the parent.
Pt_RIGHT_ANCHORED_RELATIVE	Anchor the right edge proportionally within the width of the parent.
Pt_TOP_ANCHORED_RELATIVE	Anchor the top edge proportionally within the height of the parent.
Pt_BOTTOM_ANCHORED_RELATIVE	Anchor the bottom edge proportionally within the height of the parent.
Pt_LEFT_ANCHORED_RIGHT	Anchor the left edge to the right edge of the parent.
Pt_RIGHT_ANCHORED_RIGHT	Anchor the right edge to the right edge of the parent.
Pt_TOP_ANCHORED_BOTTOM	Anchor the top edge to the bottom edge of the parent.

Constant	Description
Pt_BOTTOM_ANCHORED_BOTTOM	Anchor the bottom edge to the bottom edge of the parent.
Pt_LEFT_ANCHORED_LEFT	Anchor the left edge to the left edge of the parent.
Pt_RIGHT_ANCHORED_LEFT	Anchor the right edge to the left edge of the parent.
Pt_TOP_ANCHORED_TOP	Anchor the top edge to the top edge of the parent.
Pt_BOTTOM_ANCHORED_TOP	Anchor the bottom edge to the top edge of the parent.
Pt_BALLOONS_ON	Pop up child widget balloons immediately instead of waiting for 1.25 seconds.
Pt_ANCHORS_LOCKED	Internal informational bit.
Pt_BALLOONS_ACTIVE	Internal informational bit.
Pt_CONTAINER_RESIZING	Internal informational bit.
Pt_BALLOONS_LOCKED	Internal informational bit.
Pt_ANCHORS_INVALID	Internal informational bit.
Pt_CONTAINER_ANCHORING	Internal informational bit.

anchor_offsets

A PhRect that specifies, for each side of the widget which has its anchor_flags set, the distance between the anchored side of the widget and the corresponding side of the parent. The widget will maintain these offset distances even when the parent is resized.

This variable takes precedence over the dim, area, and resize_flags variables for this widget.

container_flags

Flags that control the behavior and appearance of the container. Default is Pt_ENABLE_CUA.

This instance variable may be a combination of zero or more of the following flags:

Constant	Description
Pt_AUTO_EXTENT	Resizes the container when children are realized, resized, or moved.
Pt_BLOCK_CUA_FOCUS	Blocks CUA focus on this container.
Pt_DISABLE_BALLOONS	Prevents use of balloons on this container and its children.
Pt_ENABLE_CUA	Permits CUA key functions in this container and its children.
Pt_ENABLE_CUA_ARROWS	Permits navigation by arrow keys in this container and its children.
Pt_HOTKEY_TERMINATOR	Blocks hotkey searches into parents of disjoint container widgets.
Pt_BALLOON_PROPAGATE	Internal informational bit.
Pt_CANVAS_INVALID	Internal informational bit.

Constant	Description
Pt_CONTAINER_FLAGS_MASK	Internal informational bit.
Pt_CURSOR_IN_CONTAINER	Internal informational bit.
Pt_IGNORE_CONSTRAINTS	Internal informational bit.
Pt_SYSINFO_VALID	Internal informational bit.

`focus`

A `PtWidget` that was the last child of this widget to have the focus.

Callbacks

The following callbacks are associated with this widget:

Callback	Description
Pt_CB_RESIZE	This callback is generated when the parent of the widget is resized.
Pt_CB_BALLOONS	This callback is generated when the pointer lingers over the specified widget for more than 1.25 seconds."
Pt_CB_FILTER	This callback is generated when an event to be passed to a child of the container matches the provided event mask.

Associated Classes

`PtContainerCallback`, `PtCallbackInfo`

PtDBContainer

PtDBContainer — A double-buffered container that eliminates screen flicker.

Synopsis

```
class PtDBContainer PtContainer
{
    db_image_type;        // integer  (Pt_ARG_DB_IMAGE_TYPE)
    memory_image_type;    // integer  (Pt_ARG_DB_MEMORY_CONTEXT_TYPE)
}
```

Base Classes

PtWidget <-- PtBasic <-- PtContainer <-- PtDBContainer

Description

This widget is a special sub-class of PtContainer that allocates extra memory (double buffer) to eliminate screen flicker for child widgets and animations.

For detailed information, please refer to PtDBContainer in the Photon documentation.

Instance Variables

db_image_type

This instance variable specifies the image format, and may have one of the following values:

Constant	Description
Pg_IMAGE_PALETTE_BYTE	An image format indexed directly to the current palette, using 1 pixel per byte, for up to 256 colors. This is the default.
Pg_IMAGE_DIRECT_888	An image format consisting of an array of 4-byte color entries, in the format [r g b x], where x is reserved.

memory_image_type

This instance variable specifies when an image is drawn to memory, and may have one of the following values:

Constant	Description
Pm_PHS_CONTEXT	Keeps the image in the PHS (Photon draw stream) as long as possible. This is the default.
Pm_IMAGE_CONTEXT	Immediately draws to the memory image.

PtDivider

PtDivider — A container with re-sizable rows or columns.

Synopsis

```
class PtDivider PtContainer
{
    divider_flags;     // flag  (Pt_ARG_DIVIDER_FLAGS)
    divider_offset;    // short  (Pt_ARG_DIVIDER_OFFSET)
    divider_sizes;     // PhPoint array  (Pt_ARG_DIVIDER_SIZES)
}
```

Base Classes

PtWidget <-- PtBasic <-- PtContainer <-- PtDivider

Description

This widget is a container that makes adjustable rows or columns out of its child widgets, such as PtButtons or PtPanes. Its instance variables let you control how or whether the children get resized, and to find their relative positions.

For detailed information, please refer to PtDivider in the Photon documentation.

Instance Variables

divider_flags

This instance variable specifies divider characteristics, and may be a combination of zero or more of the following flags:

Constant	Description
Pt_DIVIDER_NORESIZE	Generates the callback without resizing children.
Pt_DIVIDER_RESIZE_BOTH	ON, the default, resizes the widgets on either side of the handle. OFF moves the immediately right (or bottom) children and resizes the immediately left (or top) child and the far right (or bottom) child.
Pt_DIVIDER_INVISIBLE	Hides the drag outline. Used with Pt_DIVIDER_NORESIZE.
Pt_DIVIDER_CASCADE	Internal informational bit.

divider_offset

An integer specifying how much to offset the PtPoints in the divider_sizes array in both the x and y directions. This is used to modify the divider_sizes array, but has no effect on the positioning of the children of the PtDivider.

divider_sizes

A read-only array of PhPoints indicating the top left-hand corner of each child of the PtDivider.

Callbacks

The following callbacks are associated with this widget:

Callback	Description
Pt_CB_DIVIDER_DRAG	This callback is generated after each drag event the widget receives.
Pt_CB_DIVIDER_SETRESOURCES	This callback is generated when the divider resources are set.

Associated Classes

PtDividerCallback, PtCallbackInfo

Example

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

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

/*
 * This example loads a PtDivider that holds three
 * buttons arranged vertically.
 */
require_lisp("PhotonWidgets.lsp");
require_lisp("PhabTemplate.lsp");
PtInit(nil);

/* This function lets a button control the divider resize
 * feature (on or off), and makes the label on the button
 * change accordingly.
 */
function toggle_resize(button, division)
{
  if((button.flags & Pt_SET) != 0)
  {
 button.text_string = "Resizing\nis switched\nOFF";
 division.divider_flags = Pt_DIVIDER_NORESIZE;
  }
  else
  {
 button.text_string = "Resizing\nis switched\nON";
 division.divider_flags = cons(Pt_DIVIDER_NORESIZE, nil);
  }
}

wfile = PhabReadWidgetFile (string(_os_, "-WidgetFiles/wgt/divider.wgtw"));
window = PhabCreateWidgets(wfile, nil, nil);

win = PhabLookupWidget(window,#divider,nil);
div = PhabLookupWidget(window,#MainDivider,nil);
tbut = PhabLookupWidget(window,#DivButton2,nil);
exitbut = PhabLookupWidget(window,#DivButton3,nil);

div.divider_flags = Pt_DIVIDER_INVISIBLE;

PtAttachCallback(tbut, Pt_CB_ACTIVATE, ‘toggle_resize(@tbut, @div));
PtAttachCallback(exitbut, Pt_CB_ACTIVATE, #exit_program(1));

PtRealizeWidget(win);

/*
```

```
 * Assign an offset, and then print the offset and sizes.
 */
div.divider_offset = -10;
pretty_princ("Offset: ",div.divider_offset,"\n");
pretty_princ("Sizes: ",div.divider_sizes,"\n");

PtMainLoop();
```

PtEllipse

PtEllipse — An ellipse defined by a rectangular area.

Synopsis

```
class PtEllipse PtGraphic
{
}
```

Base Classes

PtWidget <-- PtBasic <-- PtGraphic <-- PtEllipse

Description

This widget is an ellipse defined by a bounding rectangle. The rectangle can be defined in one of two ways: as a dim variable, or as a points variable.

dim is set with a call to either of the SetDim or the SetArea methods. This is convenient for fast setup.

points is an array of two PhPoints. Thus, it needs to have two points defined before you can use it. However, this facilitates resizing the ellipse in code, by manipulating the points.

For other information, please refer to PtEllipse in the Photon documentation.

Example

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

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

/*
 * This example puts two PtEllipses in a window
 * and prints the dimensions of their defining
 * rectangles. The first one is defined by setting
 * its dim variable with SetArea, and the second by
 * defining points and setting its points variable.
 */
require_lisp("PhotonWidgets.lsp");
PtInit(nil);

win = new(PtWindow);
win.SetDim (200,200);
win.fill_color = 0xffffbb;

el1 = new(PtEllipse);
el1.SetArea (25,40,120,50);

pt1 = new(PhPoint);
pt2 = new(PhPoint);
pt1.x = 65;
pt1.y = 50;
pt2.x = 110;
pt2.y = 140;

el2 = new(PtEllipse);
el2.points = array(pt1,pt2);

pretty_princ("Ellipse 1: ",el1.dim,"\n");
pretty_princ("Ellipse 2: ",el2.points,"\n");

ln = new(PtLine);
```

```
o = new(PhPoint);
o.x = 50;
o.y = 150;
s = new(PhPoint);
s.x = 0;
s.y = 0;
e = new(PhPoint);
e.x = -15;
e.y = -25;

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

PtFileSel

`PtFileSel` — A tree-like widget used to select files and directories.

Synopsis

```
class PtFileSel PtGenTree
{
    fs_file_spec;    // string  (Pt_ARG_FS_FILE_SPEC)
    fs_flags;        // flag  (Pt_ARG_FS_FLAGS)
    fs_root_dir;     // string  (Pt_ARG_FS_ROOT_DIR)
}
```

Base Classes

`PtWidget` <-- `PtBasic` <-- `PtContainer` <-- `PtGenList` <-- `PtGenTree` <-- `PtFileSel`

Description

This widget reads and displays file directories in a collapsing tree format that can be expanded to show sub-directories, files, and links. It uses images to differentiate various kinds of files.

For detailed information, please refer to PtFileSel in the Photon documentation.

Instance Variables

`fs_file_spec`

A string specifying the files to match. The wildcard symbol "*" is the default. Multiple patterns can be specified; each must be separated by a comma.

`fs_flags`

This instance variable controls the characteristics of the widget, and may be a combination of zero or more of the following flags:

Constant	Description
`Pt_FS_SHOW_DIRS`	Show directories.
`Pt_FS_SHOW_FILES`	Show files.
`Pt_FS_SHOW_HIDDEN`	Show hidden directories or files.
`Pt_FS_SHOW_ERRORS`	Show files with read errors.
`Pt_FS_FREE_ON_COLLAPSE`	Frees items on every collapse.
`Pt_FS_SINGLE_LEVEL`	Use single-level mode, rather than the default tree mode.
`Pt_FS_SEEK_KEY`	In single level mode, use keyboard seek to find items.
`Pt_FS_ALL_FLAGS`	A mask containing all FS flag bits.

`fs_root_dir`

A string specifying the root directory for the file selector. Default is `nil`.

Callbacks

The following callbacks are associated with this widget:

Callback	Description
`Pt_CB_FS_STATE`	This callback is generated when an item is expanded or collapsed.
`Pt_CB_FS_SELECTION`	This callback is generated when an item is selected.
`Pt_CB_FS_BKGD_HANDLER`	This callback is generated when a directory item is read.

Associated Classes

`PtFileSelItem`, `PtFileSelCallback`, `PtFileSelBkgdCallback`, `PtCallbackInfo`

PtFolder

PtFolder — A file-folder style container.

Synopsis

```
class PtFolder PtContainer
{
    folder_panes;           // PtWidget array
    folder_selected_tab;    // short
    folder_tab_height;      // short
    folder_tabs;            // PtWidget array
}
```

Base Classes

PtWidget <-- PtBasic <-- PtContainer <-- PtFolder

Description

This widget is a folder that can contain virtual pages, which are accessed by PtFolderTabs. Each page is set up by creating a PtFolderTab, and then immediately assigning a PtPane to that tab. The PtPane and all of its children are displayed when the PtFolderTab is selected.

This widget does not appear in the Photon documentation.

Instance Variables

`folder_panes`

An array of PtPanes contained in the folder.

`folder_selected_tab`

An integer specifying the selected tab. Tabs are numbered consecutively, starting with 1. If no tab is selected, or if a tab is reselected, the value of this variable is 0.

`folder_tab_height`

A number of pixels specifying the height of the tabs for this folder. Default is 19.

`folder_tabs`

An array of PtFolderTabs contained in the folder.

Example

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

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

/* This example demonstrates PtFolder and PtFolderTab.
 * It creates a folder with three tabs.  It also includes
 * a function that de_activates the set/unset toggle
 * feature of the tabs.
 */

require_lisp ("PhotonWidgets");

/* Here is the set/unset deactivation function.
 * It keeps a tab set until another tab is selected.
 */
function do_not_unset (widget)
{
```

```
   local folder = PtWidgetParent (widget);

   if (folder.folder_selected_tab == 0)
    widget.flags = Pt_SET;
}

function main ()
{
   init_ipc ("a","a");
   PtInit (nil);

   w = new (PtWindow);
   f = new (PtFolder);
   f.SetDim (300,200);

   /* We anchor the folder to the window to demonstrate how
    * the PtPane gets resized with the folder automatically.
    */
   f.anchor_flags = cons (0, -1);
   f.anchor_flags = Pt_BOTTOM_ANCHORED_BOTTOM |
     Pt_TOP_ANCHORED_TOP |
       Pt_LEFT_ANCHORED_LEFT |
         Pt_RIGHT_ANCHORED_RIGHT;

   /* Each tab must be a child of the folder, and the
    * pane for each tab should be the first child of that
    * tab. If the pane is created immediately after the
    * tab, you don't have to set the parent explicitly.
    */
   PtSetParentWidget (f);
   tab = new (PtFolderTab);
   tab.foldertab_text_string = "Tab #1";
   tab.flags = Pt_SET;
   PtAttachCallback (tab, Pt_CB_ACTIVATE, 'do_not_unset(@tab));
   p1 = new (PtPane);
   p1.fill_color = 0xffaaaa;

   PtSetParentWidget (f);
   tab = new (PtFolderTab);
   tab.foldertab_text_string = "Tab #2";
   PtAttachCallback (tab, Pt_CB_ACTIVATE, 'do_not_unset(@tab));
   p2 = new (PtPane);
   p2.fill_color = 0x00ddaa;

   /* Note: this tab retains the set/unset toggle feature.
    */
   PtSetParentWidget (f);
   tab = new (PtFolderTab);
   tab.foldertab_text_string = "Tab #3";
   p3 = new (PtPane);
   p3.fill_color = 0xccaaff;

   PtSetParentWidget (p1);
   b = new (PtButton);
   b.text_string = "Button 1";
   b.SetPos (60,50);

   PtSetParentWidget (p2);
   b = new (PtButton);
   b.text_string = "Button 2";
   b.SetPos (120,50);

   PtSetParentWidget (p3);
   b = new (PtButton);
   b.text_string = "Button 3";
   b.SetPos (180,50);

   PtRealizeWidget (w);
```

```
   PtMainLoop();
}
```

See Also

PtFolderTab, PtPane

PtFolderTab

PtFolderTab — A tab for accessing folder contents.

Synopsis

```
class PtFolderTab PtBasic
{
    foldertab_text_font;      // string
    foldertab_text_string;    // string
}
```

Base Classes

PtWidget <-- PtBasic <-- PtFolderTab

Description

This widget is a tab that appears at the top of a PtFolder and gives access to the contents of the folder. Each PtFolderTab needs a PtPane as its first child. Clicking on the tab brings the PtPane to the front of the folder.

This widget is designed to toggle between "set" and "not set" each time it is selected. So, the first time you click on it, the folder will "open" to that tab setting. The next time you click on it, the folder will "close". If you prefer that the folder stay open to a tab no matter how many times it is clicked, you can incorporate into your code the do_not_unset function shown in the PtFolder Example.

This widget does not appear in the Photon documentation.

Instance Variables

foldertab_text_font

A string specifying the font of the tab text. Default is "helv14".

foldertab_text_string

A string comprising the text string of the tab. Default is "Tab".

Example

See the example in PtFolder.

Callbacks

The following callback is associated with this widget:

Callback	Description
Pt_CB_FOLDER_TAB_SELECTED	This callback is generated when a tab is selected.

See Also

PtFolder, PtPane

PtFontSel

PtFontSel — is used to select font attributes.

Synopsis

```
class PtFontSel PtContainer
{
    font_display;    // short  (Pt_ARG_FONT_DISPLAY)
    font_flags;      // flag  (Pt_ARG_FONT_FLAGS)
    font_name;       // string  (Pt_ARG_FONT_NAME)
    font_sample;     // PtWidget  (Pt_ARG_FONT_SAMPLE)
    font_symbol;     // long  (Pt_ARG_FONT_SYMBOL)
}
```

Base Classes

PtWidget <-- PtBasic <-- PtContainer <-- PtFontSel

Description

This widget is a dialog box that lets you choose a font, edit the size, and make it bold, italic, or anti-aliased. It also has a display for a sample text string of the selected font.

For detailed information, please refer to PtFontSel in the Photon documentation.

Instance Variables

font_display

This instance variable controls the categories of fonts offered, and may have one of the following values:

Constant	Description
Pt_FONTSEL_SCALABLE	Include scalable fonts.
Pt_FONTSEL_BITMAP	Include bitmap fonts.
Pt_FONTSEL_PROP	Include proportional fonts.
Pt_FONTSEL_FIXED	Include fixed fonts.
Pt_FONTSEL_ALL_FONTS	Include all fonts.

font_flags

This instance variable controls characteristics of the widget, and may be a combination of zero or more of the following flags:

Constant	Description
Pt_FONTSEL_AA_CHECK	Restricts the use of anti-aliasing button to scalable fonts.
Pt_FONTSEL_SAMPLE	Displays sample text using the current font.

font_name

A string specifying the initial and/or currently selected font. Default is "helv12".

`font_sample`

A string specifying the sample to be displayed if the `Pt_FONTSEL_SAMPLE` flag is set. Default is `"AaBbCcXxYyZz"`.

`font_symbol`

A character specifying the font family to include in the selection dialog. 'A', the default, specifies Latin fonts.

Callbacks

The following callback is associated with this widget:

Callback	Description
`Pt_CB_FONT_MODIFY`	This callback is generated when a selected font is modified.

Associated Classes

`PtCallbackInfo`

PtGauge

PtGauge — A parent class for gauge widget resources.

Synopsis

```
class PtGauge PtBasic
{
    gauge_flags;           // flag  (Pt_ARG_GAUGE_FLAGS)
    gauge_font;            // string  (Pt_ARG_GAUGE_FONT)
    gauge_h_align;         // unsigned char  (Pt_ARG_GAUGE_H_ALIGN)
    gauge_maximum;         // long  (Pt_ARG_GAUGE_MAXIMUM)
    gauge_minimum;         // long  (Pt_ARG_GAUGE_MINIMUM)
    gauge_orientation;     // char  (Pt_ARG_GAUGE_ORIENTATION)
    gauge_v_align;         // unsigned char  (Pt_ARG_GAUGE_V_ALIGN)
    gauge_value;           // long  (Pt_ARG_GAUGE_VALUE)
    gauge_value_prefix;    // string  (Pt_ARG_GAUGE_VALUE_PREFIX)
    gauge_value_suffix;    // string  (Pt_ARG_GAUGE_VALUE_SUFFIX)
}
```

Base Classes

PtWidget <-- PtBasic <-- PtGauge

Derived Classes

PtScale, PtSlider, RtProgress

Description

This class serves as a parent class of resources for gauge widgets, and is not normally instantiated.

For detailed information, please refer to PtGauge in the Photon documentation.

Instance Variables

gauge_flags

This instance variable controls characteristics of the gauge display, and may have one of the following values:

Constant	Description
Pt_GAUGE_MAX_ON_TOP	Position the maximum value on the top.
Pt_GAUGE_MAX_ON_BOTTOM	Position the maximum value on the bottom.
Pt_GAUGE_MAX_ON_LEFT	Position the maximum value on the left.
Pt_GAUGE_MAX_ON_RIGHT	Position the maximum value on the right.

In addition, it may have zero or more of the following values:

Constant	Description
Pt_SHOW_VALUE	Display the gauge value. This flag must be set in order to access the gauge_h_align or gauge_v_align variables.
Pt_VALUE_XOR	Invert the display and the background.

`gauge_font`

A string specifying the font to use for the gauge value, title, and any text. Default is `"helv12"`.

`gauge_h_align`

This instance variable controls horizontal alignment of the value display. It requires the constant `Pt_SHOW_VALUE` to be set in `gauge_flags`, and only works with gauges that support it.

It may have one of the following values:

Constant	Description
`Pt_LEFT`	Align the value display to the left edge.
`Pt_RIGHT`	Align the value display to the right edge.
`Pt_CENTER`	Center the value display horizontally.

`gauge_maximum`

A number specifying the gauge's maximum value.

`gauge_minimum`

A number specifying the gauge's minimum value.

`gauge_orientation`

This instance variable specifies the axis for drawing the gauge, and may have one of the following values:

Constant	Description
`Pt_VERTICAL`	Draw gauge on vertical axis.
`Pt_HORIZONTAL`	Draw gauge on horizontal axis.

`gauge_v_align`

This instance variable controls vertical alignment of the value display. It requires the constant `Pt_SHOW_VALUE` to be set in `gauge_flags`, and only works with gauges that support it.

It may have one of the following values:

Constant	Description
`Pt_TOP`	Align the value display to the top.
`Pt_BOTTOM`	Align the value display to the bottom.
`Pt_CENTER`	Center the value display vertically.

`gauge_value`

A number specifying the current value of the gauge.

`gauge_value_prefix`

A string that is attached to and displayed before the value of the gauge. For example, the string "Tank Level: " would give a display of `Tank Level: 155`.

`gauge_value_suffix`

A string that is attached to and displayed after the value of the gauge. For example, the string " cm" would give a display of `155 cm`.

PtGenList

PtGenList — A parent class for list and tree widget resources.

Synopsis

```
class PtGenList PtContainer
{
    balloon_color;          // color  (Pt_ARG_BALLOON_COLOR)
    balloon_fill_color;     // color  (Pt_ARG_BALLOON_FILL_COLOR)
    list_flags;             // flag  (Pt_ARG_LIST_FLAGS)
    list_font;              // string  (Pt_ARG_LIST_FONT)
    list_item_count;        // unsigned short  (Pt_ARG_LIST_ITEM_COUNT)
    list_scroll_rate;       // short  (Pt_ARG_LIST_SCROLL_RATE)
    list_sel_count;         // unsigned short  (Pt_ARG_LIST_SEL_COUNT)
    list_total_height;      // unsigned short  (Pt_ARG_LIST_TOTAL_HEIGHT)
    scrollbar_width;        // unsigned short  (Pt_ARG_SCROLLBAR_WIDTH)
    selection_fill_color;   // color  (Pt_ARG_SELECTION_FILL_COLOR)
    selection_mode;         // unsigned short  (Pt_ARG_SELECTION_MODE)
    selection_text_color;   // color  (Pt_ARG_SELECTION_TEXT_COLOR)
    top_item_pos;           // unsigned short  (Pt_ARG_TOP_ITEM_POS)
    visible_count;          // unsigned short  (Pt_ARG_VISIBLE_COUNT)
}
```

Base Classes

PtWidget <-- PtBasic <-- PtContainer <-- PtGenList

Derived Classes

PtGenTree, PtList

Description

This class serves as a parent class of resources for list widgets, and is not normally instantiated.

For detailed information, please refer to PtGenList in the Photon documentation.

Instance Variables

`balloon_color`

A number specifying the balloon's text color. Default is `0xc814` (green).

`balloon_fill_color`

A number specifying the balloon's background fill color. Default is `0xc814` (green).

`list_flags`

This instance variable controls the appearance and behavior of the list, and may be a combination of zero or more of the following flags:

Constant	Description
Pt_LIST_SCROLLBAR_NEVER	Scrollbar Never
Pt_LIST_SCROLLBAR_ALWAYS	A scrollbar is always displayed.
Pt_LIST_SCROLLBAR_AS_REQUIRED	A scrollbar is only displayed when required.
Pt_LIST_INACTIVE	Inactivates the list.
Pt_LIST_NON_SELECT	Restricts the list to read-only.
Pt_LIST_SNAP	Snaps the list to fit the number of items.

Constant	Description
Pt_LIST_BALLOON_NEVER	Internal informational bit.
Pt_LIST_SHOW_BALLOON	Shows list balloons.
Pt_LIST_BALLOON_AS_REQUIRED	Shows list balloons if list is clipped.
Pt_LIST_BALLOON_REGISTERED	Internal informational bit.
Pt_LIST_SCROLLBAR_GETS_FOCUS	Gives focus to the scrollbar.
Pt_LIST_NOBLIT	Prevents flushing and blitting when Pt_ARG_TOP_ITEM_POS changes.

`list_font`

A string specifying the font used for list items. Default is `"helv12"`.

`list_item_count`

A number specifying the number of items in the list.

`list_scroll_rate`

A number specifying the rate of scrolling. Default is `2`. Set it to `1` to scroll faster, set it higher to scroll slower.

`list_sel_count`

A read-only number of list items.

`list_total_height`

A read-only number indicating the total height in pixels of all list items.

`scrollbar_width`

A number of pixels specifying the width of the scrollbar. Default is `15`. When set to `0`, the default is used as well.

`selection_fill_color`

A number specifying the fill color for selected items. Default is `0xff` (blue).

`selection_mode`

A constant specifying the way items are chosen from the list, with mouse or keyboard. If a mouse is not used, the **Up** and **Down** arrows on the numeric keypad move the selection highlight, and pressing the **Enter** key makes the selection.

This instance variable may have one of the following values:

Constant	Description
Pt_SINGLE_MODE	Allows selection of one item by clicking on it.
Pt_MULTIPLE_MODE	Allows selection of multiple items by clicking on them. A second click releases a selected item.
Pt_BROWSE_MODE	Allows selection of one item by clicking on it, or dragging down and releasing. This is the default.
Pt_EXTENDED_MODE	Allows selection of a range of items with **Shift**-click, or multiple disjoint items with **Ctrl**-click.

Constant	Description
Pt_RANGE_MODE	Allows selection of a range of items by dragging from the first to the last; the list will scroll.

In addition to the constant above, this instance variable may also have zero or more of the following flags:

Constant	Description
Pt_SELECTION_MODE_MULTIPLE	Allows multiple items to be selected independently.
Pt_SELECTION_MODE_RANGE	Allows selection of a range of items.
Pt_SELECTION_MODE_SINGLE	Allows selection of a single item.
Pt_SELECTION_MODE_NONE	Selection is done by callback.
Pt_SELECTION_MODE_NOSCROLL	Prevents scrolling during drag operations.
Pt_SELECTION_MODE_NOMOVE	Holds the item still during drag operations.
Pt_SELECTION_MODE_NOREST	Maintains the existing state if the mouse button is released outside the widget.
Pt_SELECTION_MODE_AUTO	Selection is done by keyboard, unless the **Ctrl** key is used.
Pt_SELECTION_MODE_NOCLEAR	In Pt_SELECTION_MODE, doesn't clear existing selection when new selection is made.
Pt_SELECTION_MODE_TOGGLE	For Pt_SINGLE_MODE and Pt_MULTIPLE_MODE, mouse clicks toggle select/unselect.
Pt_SELECTION_MODE_NOFOCUS	For Pt_SELECTION_MODE_AUTO, the current item is not automatically selected when the widget gets focus.
Pt_SELECTION_MODE_COMPOSE_FLAG	Internal informational bit.
Pt_SELECTION_MODE_COMPOSE_MASK	Internal informational bit.

`selection_text_color`

A number specifying the color of text for selected items. Default is `0xffffff` (white).

`top_item_pos`

An integer specifying the index number of the item to appear at the top of the list. (Items are numbered consecutively, starting at one.) Default is `1`, the first item.

`visible_count`

A read-only number indicating the number of currently visible items. Default is `0`.

PtGenTree

PtGenTree — A parent class for tree widget resources.

Synopsis

```
class PtGenTree PtGenList
{
    tree_flags;    // flag  (Pt_ARG_TREE_FLAGS)
}
```

Base Classes

PtWidget <-- PtBasic <-- PtContainer <-- PtGenList <-- PtGenTree

Derived Classes

PtFileSel, PtTree

Description

This class serves as a parent class of resources for tree widgets, and is not normally instantiated. Tree widgets are lists, each item of which can be expanded to display an attached sub-list, or collapsed to hide the sub-list. The items at the first level of the tree are called root items, and are always visible.

Trees can be built independently and added as roots or trees to the main tree. Items can be added separately as well, but this requires calls to PtHold and PtUpdate functions to avoid multiple redraws.

For detailed information, please refer to PtGenTree in the Photon documentation.

Instance Variables

tree_flags

This instance variable controls the appearance and behavior of the tree, and may be a combination of zero or more of the following flags:

Constant	Description
Pt_TREE_HAS_BUTTONS	Display the + and - buttons.
Pt_TREE_HAS_LINES	Display connecting lines.
Pt_TREE_ROOT_LINES	Display root lines and buttons, if any.
Pt_TREE_TO_RIGHT	Items are extended to right edge.
Pt_TREE_TO_LEFT	Background is extended to left edge.
Pt_TREE_COLLAPSING	A subtype of Pt_CB_TREE_STATE, indicating a collapsing state.
Pt_TREE_EXPANDING	A subtype of Pt_CB_TREE_STATE, indicating an expanding state.
Pt_TREE_ITEM_EXPANDABLE	The item can be expanded.
Pt_TREE_ITEM_EXPANDED	The item branches are expanded and displayed.

Callbacks

The following callback is associated with this widget:

Callback	Description
Pt_CB_GEN_TREE_INPUT	This callback is generated from mouse and key events.

Associated Classes

PtGenTreeInput, PtCallbackInfo

PtGraphic

PtGraphic — A parent class for graphic widget resources.

Synopsis

```
class PtGraphic PtBasic
{
    dash_list;        // char array  (Pt_ARG_DASH_LIST)
    dash_scale;       // long  (Pt_ARG_DASH_SCALE)
    graphic_flags;    // flag  (Pt_ARG_GRAPHIC_FLAGS)
    line_cap;         // unsigned short  (Pt_ARG_LINE_CAP)
    line_join;        // unsigned short  (Pt_ARG_LINE_JOIN)
    line_width;       // long  (Pt_ARG_LINE_WIDTH)
    origin;           // PhPoint  (Pt_ARG_ORIGIN)
    points;           // PhPoint array  (Pt_ARG_POINTS)
}
```

Base Classes

PtWidget <-- PtBasic <-- PtGraphic

Derived Classes

PtArc, PtBezier, PtEllipse, PtLine, PtPixel, PtPolygon, PtRect

Description

This widget serves as a parent class of resources for graphic widgets, and is not normally instantiated. The variables here have to do mainly with line origins, thicknesses, joints and caps, as well as fill colors for closed shapes.

For detailed information, please refer to PtGraphic in the Photon documentation.

Instance Variables

`dash_list`

An array of integers specifying in pixels the length of each dash and/or space for dashed lines. If there is more than one element in the array, each element applies to the next dash or space, and then the pattern repeats. An array with an odd number of elements will produce an alternating pattern of dashes and spaces (see the example below). The default is an empty array `[ ]`, which gives a solid line.

Setting this variable to an array of length `0` with a Gamma statement like this: `GraphicInstance.dash_list = array(0);` will cause undefined behavior. To reset the default, use `array()`.

`dash_scale`

A 32 bit number that specifies the scale of the `dash_list`. The top 16 bits correspond to numbers greater than one, the bottom 16 to fractions less than one. The number 0x00010000 is equal to one.

For example, to scale dashes 2 times larger, you would use the number 0x00020000; to make them half size, you would use 0x00008000 (because that is half of 0x00010000). See also the example below.

graphic_flags

This instance variable controls the position and origin of the widget and the graphic with respect to the parent widget. It may be a combination of zero (the default) or more of the following flags:

Constant	Description
Pt_FLOAT_POS	Allow the position and origin of the widget to change, leaving the graphic fixed to the parent.
Pt_FLOAT_ORIGIN	Allow the origin of the graphic to change, leaving the position of the widget fixed to the parent.

line_cap

A constant that determines the shape of line ends, which become visible when a line is fairly wide. The default is butted, which means the line ends squarely at the end point. You can put a round or square cap at the end, which overlaps the end point by a factor of half the line width.

This instance variable may have one of the following values:

Constant	Description
Pg_BUTT_CAP	The default, leaves line capless, with square-cut ends.
Pg_ROUND_CAP	Puts a round cap on thick lines.
Pg_SQUARE_CAP	Puts a square cap on thick lines.

line_join

This instance variable determines the shape of line joints (mainly visible for wide lines), and may have one of the following values:

Constant	Description
Pg_MITER_JOIN	Makes a squared-off joint.
Pg_ROUND_JOIN	Makes a small-radius rounded joint.
Pg_BEVEL_JOIN	Makes a beveled joint.
Pg_QROUND_JOIN	Makes a large-radius rounded joint.
Pg_BUTT_JOIN	Makes a butt joint (outside line edges don't join).

line_width

A number specifying the width of a line, in pixels.

origin

A PhPoint specifying the origin of the widget, which is the upper-left corner of its widget canvas.

points

An array of PhPoints defining the graphic. The number and purpose of the points depends on the type of graphic.

Example

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

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

/*
 * This example illustrates dash variables for graphic lines.
 */
require_lisp("PhotonWidgets.lsp");
require_lisp("PhabTemplate.lsp");
PtInit(nil);

wfile = PhabReadWidgetFile (string(_os_, "-WidgetFiles/wgt/graphics.wgtw"));
window = PhabCreateWidgets(wfile, nil, nil);

win = PhabLookupWidget(window,#graphics,nil);
ln0 = PhabLookupWidget(window,#Line0,nil);
ln1 = PhabLookupWidget(window,#Line1,nil);
ln2 = PhabLookupWidget(window,#Line2,nil);
ln3 = PhabLookupWidget(window,#Line3,nil);
ln4 = PhabLookupWidget(window,#Line4,nil);
ln5 = PhabLookupWidget(window,#Line5,nil);

ebut = PhabLookupWidget(window,#ExitButton,nil);
PtAttachCallback(ebut, Pt_CB_ACTIVATE, #exit_program(1));

ln0.dash_list = array();
ln1.dash_list = array(20);
ln2.dash_list = array(20,5);
ln3.dash_list = array(20,5,20);

ln4.dash_list = array(20);
ln4.dash_scale = 0x00020000;

ln5.dash_list = array(20);
ln5.dash_scale = 0x00004000;

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

Callbacks

The following callback is associated with this widget:

Callback	Description
`Pt_CB_RESCALE`	This callback is generated when the widget's dim or area resources are modified.

Associated Classes

`PtCallbackInfo`

PtGrid

PtGrid — A grid of horizontal and vertical lines.

Synopsis

```
class PtGrid PtBasic
{
    dash_list;          // char array  (Pt_ARG_DASH_LIST)
    dash_scale;         // long  (Pt_ARG_DASH_SCALE)
    grid_horizontal;    // unsigned short  (Pt_ARG_GRID_HORIZONTAL)
    grid_vertical;      // unsigned short  (Pt_ARG_GRID_VERTICAL)
    line_cap;           // unsigned short  (Pt_ARG_LINE_CAP)
    line_join;          // unsigned short  (Pt_ARG_LINE_JOIN)
    line_width;         // long  (Pt_ARG_LINE_WIDTH)
}
```

Base Classes

PtWidget <-- PtBasic <-- PtGrid

Description

This widget is a grid of horizontal and vertical lines.

For detailed information, please refer to PtGrid in the Photon documentation.

Instance Variables

dash_list

This variable is identical to the dash_list variable of PtGraphic widget.

dash_scale

This variable is identical to the dash_scale variable of PtGraphic widget.

grid_horizontal

An integer specifying the number of horizontal grid lines. Default is 4.

grid_vertical

An integer specifying the number of vertical grid lines. Default is 4.

line_cap

This instance variable determines the shape of line ends (mainly visible for wide lines), and may have one of the following values:

Constant	Description
Pg_BUTT_CAP	The default, leaves line capless, with square-cut ends.
Pg_ROUND_CAP	Puts on a round cap.
Pg_SQUARE_CAP	Puts on a square cap.

line_join

This instance variable determines the shape of line joints (mainly visible for wide lines), and may have one of the following values:

Constant	Description
Pg_MITER_JOIN	Makes a squared-off joint.
Pg_ROUND_JOIN	Makes a small-radius rounded joint.
Pg_BEVEL_JOIN	Makes a beveled joint.
Pg_QROUND_JOIN	Makes a large-radius rounded joint.
Pg_BUTT_JOIN	Makes a butt joint (outside line edges don't join).

`line_width`

A number specifying the width of the grid lines, in pixels.

Example

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

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

/*
 * This example demonstrates a PtGrid.
 */

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

win = new(PtWindow);
grid = new(PtGrid);

grid.SetDim(350,350);
grid.fill_color = 0xccddff;
grid.grid_vertical = 12;
grid.grid_horizontal = 20;
grid.line_width = 1;


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

PtGroup

PtGroup — groups widgets and manages their geometry.

Synopsis

```
class PtGroup PtContainer
{
    group_flags;          // flag  (Pt_ARG_GROUP_FLAGS)
    group_horz_align;     // unsigned short  (Pt_ARG_GROUP_HORZ_ALIGN)
    group_orientation;    // unsigned short  (Pt_ARG_GROUP_ORIENTATION)
    group_rows_cols;      // unsigned short  (Pt_ARG_GROUP_ROWS_COLS)
    group_spacing;        // unsigned short  (Pt_ARG_GROUP_SPACING)
    group_vert_align;     // unsigned short  (Pt_ARG_GROUP_VERT_ALIGN)
}
```

Base Classes

PtWidget <-- PtBasic <-- PtContainer <-- PtGroup

Derived Classes

PtMenu, PtMenuBar

Description

This widget is an invisible container that groups widgets into columns, rows, or a matrix, and manages their geometry. It permits exclusive or group selection of widgets.

For detailed information, please refer to PtGroup in the Photon documentation.

Instance Variables

group_flags

This instance variable controls the behavior of the group, and may be a combination of zero or more of the following flags:

Constant	Description
Pt_GROUP_EXCLUSIVE	Only one child can be selected at a time.
Pt_GROUP_EQUAL_SIZE	All children are forced to be the same height and width.
Pt_GROUP_NO_SELECT_ALLOWED	Allows unselecting a child (click on it) without selecting another child.
Pt_GROUP_NO_KEYS	Blocks any group use of keys.
Pt_GROUP_NO_KEY_WRAP_HORIZONTAL	Blocks keys for horizontal wrap.
Pt_GROUP_NO_KEY_WRAP_VERTICAL	Blocks keys for vertical wrap.
Pt_GROUP_NO_KEY_WRAP	Blocks keys for horizontal and vertical wrap.
 Among the next 4 flags, you should not set EQUAL_SIZE and STRETCH flags for the same direction.	
Pt_GROUP_EQUAL_SIZE_HORIZONTAL	Keeps all children in the group the same width.
Pt_GROUP_EQUAL_SIZE_VERTICAL	Keeps all children in the group the same height.
Pt_GROUP_STRETCH_HORIZONTAL	Fills the group canvas horizontally by stretching right-hand children.

Pt_GROUP_STRETCH_VERTICAL	Fills the group canvas vertically by stretching bottom children.
Pt_GROUP_STRETCH	Fills the canvas in the orientation direction by stretching the widgets of that edge.
Pt_GROUP_STRETCH_FILL	Stretch Fill

group_horz_align

This instance variable specifies horizontal alignment of the group as a whole, and may have one of the following values:

Constant	Description
Pt_GROUP_HORZ_NONE	No horizontal repositioning for the group.
Pt_GROUP_HORZ_CENTER	The group is centered horizontally.
Pt_GROUP_HORZ_LEFT	The group is aligned to the left.
Pt_GROUP_HORZ_RIGHT	The group is aligned to the right.

group_orientation

This instance variable re-aligns child widgets into rows or columns, and may have one of the following values:

Constant	Description
Pt_GROUP_HORIZONTAL	Children of group are aligned in a row.
Pt_GROUP_VERTICAL	Children of group are aligned in a column.
Pt_GROUP_ASIS	Children of group are not aligned.

group_rows_cols

A number specifying the number of rows or columns that the children are arranged into, for the relevant group_orientation chosen.

group_spacing

A number specifying the number of pixels to separate each child of the group in both the x and y directions.

group_vert_align

This instance variable specifies vertical alignment of the group as a whole, and may have one of the following values:

Constant	Description
Pt_GROUP_VERT_NONE	No vertical repositioning of the group.
Pt_GROUP_VERT_CENTER	The group is centered vertically.
Pt_GROUP_VERT_TOP	The group is aligned to the top.
Pt_GROUP_VERT_BOTTOM	The group is aligned to the bottom.

PtHtml

PtHtml — An HTML viewer.

Synopsis

```
class PtHtml PtContainer
{
    html_border_width;          // unsigned short  (Pt_ARG_HTML_BORDER_WIDTH)
    html_cursor_busy;           // unsigned short  (Pt_ARG_HTML_CURSOR_BUSY)
    html_cursor_default;        // unsigned short  (Pt_ARG_HTML_CURSOR_DEFAULT)
    html_cursor_link;           // unsigned short  (Pt_ARG_HTML_CURSOR_LINK)
    html_fill_color;            // color  (Pt_ARG_HTML_FILL_COLOR)
    html_flags;                 // flag  (Pt_ARG_HTML_FLAGS)
    html_h1_font;               // string  (Pt_ARG_HTML_H1_FONT)
    html_h2_font;               // string  (Pt_ARG_HTML_H2_FONT)
    html_h3_font;               // string  (Pt_ARG_HTML_H3_FONT)
    html_h4_font;               // string  (Pt_ARG_HTML_H4_FONT)
    html_h5_font;               // string  (Pt_ARG_HTML_H5_FONT)
    html_h6_font;               // string  (Pt_ARG_HTML_H6_FONT)
    html_link_color;            // color  (Pt_ARG_HTML_LINK_COLOR)
    html_page_bm;               // unsigned short  (Pt_ARG_HTML_PAGE_BM)
    html_page_h;                // integer  (Pt_ARG_HTML_PAGE_H)
    html_page_lm;               // unsigned short  (Pt_ARG_HTML_PAGE_LM)
    html_page_rm;               // unsigned short  (Pt_ARG_HTML_PAGE_RM)
    html_page_tm;               // unsigned short  (Pt_ARG_HTML_PAGE_TM)
    html_page_w;                // integer  (Pt_ARG_HTML_PAGE_W)
    html_page_x;                // integer  (Pt_ARG_HTML_PAGE_X)
    html_page_y;                // integer  (Pt_ARG_HTML_PAGE_Y)
    html_scroll_color;          // color  (Pt_ARG_HTML_SCROLL_COLOR)
    html_scroll_fill_color;     // color  (Pt_ARG_HTML_SCROLL_FILL_COLOR)
    html_scroll_horizontal;     // unsigned short  (Pt_ARG_HTML_SCROLL_HORIZONTAL)
    html_scroll_vertical;       // unsigned short  (Pt_ARG_HTML_SCROLL_VERTICAL)
    html_scroll_width;          // unsigned short  (Pt_ARG_HTML_SCROLL_WIDTH)
    html_text_font;             // string  (Pt_ARG_HTML_TEXT_FONT)
    html_url;                   // string  (Pt_ARG_HTML_URL)
}
```

Base Classes

PtWidget <-- PtBasic <-- PtContainer <-- PtHtml

Description

This widget is a display viewer for HTML files, with horizontal and vertical scrollbars. Designed for creating helpviewers and similar applications, it supports only HTML 1.0, plus simple tables and a few other features. It loads and displays HTML pages from the local file system.

For detailed information, please refer to PtHtml in the Photon documentation.

Instance Variables

html_border_width

The width of an internal etched border on the page and scrollbars.

html_cursor_busy

One of the following constants specifying the bitmap image for the cursor when loading a file:

Ph_CURSOR_POINTER
Ph_CURSOR_BIG_POINTER
Ph_CURSOR_CROSSHAIR
Ph_CURSOR_FINGER

Ph_CURSOR_CLOCK (the default)

`html_cursor_default`

One of the following constants specifying the normal bitmap image for the cursor:

Ph_CURSOR_POINTER (the default)
Ph_CURSOR_BIG_POINTER
Ph_CURSOR_CROSSHAIR
Ph_CURSOR_FINGER

`html_cursor_link`

One of the following constants specifying the bitmap image for the cursor when over a link:

Ph_CURSOR_POINTER
Ph_CURSOR_BIG_POINTER
Ph_CURSOR_CROSSHAIR
Ph_CURSOR_FINGER (the default)

`html_fill_color`

A number specifying a color for the background of the display. Default is `0xc0c0c0` (grey).

`html_flags`

This instance variable determines the behavior of the widget, and may have one of the following values:

Constant	Description
`Pt_HTML_RELOAD`	Reloads a page.
`Pt_HTML_PAGE_MODE`	Paginates the page for printing.

`html_h1_font`

A string specifying the font for a level 1 heading. Default is `"helv24"`.

`html_h2_font`

Level 2 heading font. Default is `"helv18"`.

`html_h3_font`

Level 3 heading font. Default is `"helv14"`.

`html_h4_font`

Level 4 heading font. Default is `"helv12"`.

`html_h5_font`

Level 5 heading font. Default is `"helv10"`.

`html_h6_font`

Level 6 heading font. Default is `"helv08"`.

`html_link_color`

A number specifying the color for links. Default is `0xa0` (dark blue).

`html_page_bm`, html_page_lm, html_page_rm, html_page_tm

Integers specifying the width in pixels of the bottom, left, right and top margins of the page. Default for each is 5.

`html_page_h`, html_page_w

Integers indicating the height and width in pixels of the displayed HTML page.

`html_page_x`, html_page_y

Integers specifying in pixels the horizontal and vertical position of the viewing area with respect to the page. This number changes when the page is scrolled.

`html_scroll_color`

An integer specifying the color of the handle of the scrollbar. Default is `0xc0c0c0` (grey).

`html_scroll_fill_color`

An integer specifying the color of the trough of the scrollbar. Default is `0x909090` (dark grey).

`html_scroll_horizontal`, html_scroll_vertical

These instance variables specify the mode of the horizontal and vertical scrollbars. Each may have one of the following values:

Constant	Description
`Pt_ALWAYS`	The scrollbar is always displayed (the default).
`Pt_AS_REQUIRED`	The scrollbar is displayed only when the page is wider than the viewer.
`Pt_ALWAYS`	The scrollbar is never displayed.

`html_scroll_width`

A number specifying the width in pixels of the scrollbars. Default is `15`.

`html_text_font`

A string specifying the body text font. Default is "helv14".

`html_url`

The URL of the HTML page to display. The path can be relative or absolute, and can include a node and directory.

Callbacks

The following callbacks are associated with this widget:

Callback	Description
`Pt_CB_HTML_FILE_PRE`	This callback is generated when an HTML file is going to be loaded.
`Pt_CB_HTML_FILE_POST`	This callback is generated when an HTML file has been loaded.
`Pt_CB_HTML_IMAGE`	This callback is generated when an HTML image file is going to be loaded.

Associated Classes

`PtHtmlCallback`, `PtCallbackInfo`

Example

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

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

/*
 * This example demonstrates a PtHtml widget.
 */

if (_os_ == "QNX6")
     princ ("\nThis widget is not available for QNX 6.", "\n\n");
else
{
  require_lisp("PhotonWidgets.lsp");
  PtInit(nil);

  win = new(PtWindow);
  html = new(PtHtml);

  html.SetDim(700,350);
  html.html_url = "re-pthtml.html";

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

PtIcon

PtIcon — A container for a Photon Desktop Manager icon.

Synopsis

```
class PtIcon PtWindow
{
}
```

Base Classes

PtWidget <-- PtBasic <-- PtContainer <-- PtWindow <-- PtIcon

Description

This widget is a miniature window that holds icons. It allows two sizes: 43 X 43 pixels for the Photon Desktop Manager, and 15 X 15 pixels for the Taskbar.

For detailed information, please refer to PtIcon in the Photon documentation.

PtLabel

PtLabel — A label that displays text, bitmaps, or images.

Synopsis

```
class PtLabel PtBasic
{
    accel_key;              // string  (Pt_ARG_ACCEL_KEY)
    balloon_color;          // color  (Pt_ARG_BALLOON_COLOR)
    balloon_fill_color;     // color  (Pt_ARG_BALLOON_FILL_COLOR)
    balloon_position;       // short  (Pt_ARG_BALLOON_POSITION)
    horizontal_alignment;   // unsigned char  (Pt_ARG_HORIZONTAL_ALIGNMENT)
    label_data;             // PhImage  (Pt_ARG_LABEL_DATA)
    label_flags;            // flag  (Pt_ARG_LABEL_FLAGS)
    label_type;             // char  (Pt_ARG_LABEL_TYPE)
    line_spacing;           // unsigned short  (Pt_ARG_LINE_SPACING)
    margin_bottom;          // unsigned short  (Pt_ARG_MARGIN_BOTTOM)
    margin_left;            // unsigned short  (Pt_ARG_MARGIN_LEFT)
    margin_right;           // unsigned short  (Pt_ARG_MARGIN_RIGHT)
    margin_top;             // unsigned short  (Pt_ARG_MARGIN_TOP)
    select_shift;           // unsigned short  (Pt_ARG_SELECT_SHIFT)
    text_font;              // string  (Pt_ARG_TEXT_FONT)
    text_string;            // string  (Pt_ARG_TEXT_STRING)
    underline1;             // color  (Pt_ARG_UNDERLINE1)
    underline2;             // color  (Pt_ARG_UNDERLINE2)
    underline_type;         // unsigned short  (Pt_ARG_UNDERLINE_TYPE)
    vertical_alignment;     // unsigned char  (Pt_ARG_VERTICAL_ALIGNMENT)
}
```

Base Classes

PtWidget <-- PtBasic <-- PtLabel

Derived Classes

PtButton, PtMenuLabel, PtText, PtToggleButton

Description

This widget is a label that can hold text, bitmaps, or images, and has an optional pop-up text balloon. Labels can be used to identify widgets and data entry fields, make icons, or put pictures into your applications.

For detailed information, please refer to PtLabel in the Photon documentation.

Instance Variables

accel_key

A string specifying a character to use as an accelerator key or hot key.

balloon_color

A number specifying the color for pop-up balloon text. Default is 0x0 (black).

balloon_fill_color

A number specifying the color for the pop-up balloon background. Default is 0xfeffb1 (yellow).

balloon_position

This instance variable specifies the position of the pop-up balloon, and may have one of the following values:

Constant	Description
Pt_BALLOON_RIGHT	Right
Pt_BALLOON_LEFT	Left
Pt_BALLOON_TOP	Top
Pt_BALLOON_BOTTOM	Bottom
Pt_BALLOON_INPLACE	In Place

horizontal_alignment

This instance variable specifies the horizontal alignment of the label text, and may have one of the following values:

Constant	Description
Pt_RIGHT	Align the label text to the right.
Pt_LEFT	Align the label text to the left.
Pt_CENTER	Center the label text horizontally.

label_data

A PhImage displayed by the label if the label_type is set to Pt_IMAGE.

label_flags

This instance variable controls the appearance and behavior of the widget, and may be a combination of zero or more of the following flags:

Constant	Description
Pt_LABEL_SELECT_SHIFT	Cause the content of the widget to shift down and right when the widget is selected.
Pt_SHOW_BALLOON	Cause balloon help to pop up on the widget if the cursor is left stationary over the widget for 1.25 seconds.
Pt_BALLOON_AS_REQUIRED	Only show balloon help if it is different from the visible content of the widget.
Pt_BACKFILL_TEXT	Cause the widget to fill the text rectangle with the background color before rendering the text.
Pt_BALLOON_REGISTERED	Internal informational bit.

label_type

This instance variable specifies the type of label, and may have one of the following values:

Constant	Description

Constant	Description
Pt_Z_STRING	Display text using the text_string variable.
Pt_BITMAP	Bitmap
Pt_IMAGE	Display a PhImage using the label_data variable.
Pt_TEXT_IMAGE	Display text and an image, positioned by the balloon_position variable.

line_spacing

A number of pixels to put between lines of text for extra space.

margin_bottom, margin_left, margin_right, margin_top

A number of pixels on the bottom, left, right, and top between the text and the edge of the label for margins.

select_shift

This instance variable is deprecated. Set the Pt_LABEL_SELECT_SHIFT flag in label_flags instead.

text_font

A string specifying the font used for label text.

text_string

The string used for the label text string if the label_type is set to Pt_Z_STRING or Pt_TEXT_IMAGE.

underline1

A number specifying the color for the first underline. Default is 0x0 (black).

underline2

A number specifying the color for the second underline, which is just below the first underline. Making the two the same color creates a thick underline. Default is 0xffffffff (transparent).

underline_type

A constant that specifies the type of underline for the text. For single or double underlines, the underline colors are specified using underline1 and underline2 (above). Etched underlines use the colors of the widget's top_border_color and bot_border_color.

This instance variable may have one of the following values:

Constant	Description
Pt_NO_ULINE	Text is not underlined.
Pt_SINGLE_ULINE	Text gets a singe underline.
Pt_DOUBLE_ULINE	Text gets a double underline.
Pt_ULINE_ETCHED_IN	The underline gets an 'etched-in' appearance.
Pt_ULINE_ETCHED_OUT	The underline gets an 'etched-out' appearance.

`vertical_alignment`

This instance variable determines the vertical alignment of the text string, and may have one of the following values:

Constant	Description
`Pt_BOTTOM`	Align the label text to the bottom.
`Pt_TOP`	Align the label text to the top.
`Pt_CENTER`	Center the label text vertically.

PtLed

PtLed — An LED-style data display.

Synopsis

```
class PtLed PtBasic
{
    led_data;         // short array
    led_depth;        // short
    led_flags;        // flag
    led_max;          // short
    led_min;          // short
    led_off_color;    // color array
    led_on_color;     // color array
    led_rows;         // short
    led_segs;         // short
    led_space;        // short
}
```

Base Classes

PtWidget <-- PtBasic <-- PtLed

Derived Classes

PtLedBar, PtLedPanel

Description

This widget is not yet implemented, and does not appear in the Photon documentation.

Instance Variables

led_data

led_depth

led_flags

This instance variable may be a combination of zero or more of the following flags:

Constant	Description
Pt_LED_HORZ	Horizontal
Pt_LED_VERT	Vertical
Pt_LED_HIGHLIGHT	Highlight
Pt_LED_SET	Set

led_max

led_min

`led_off_color`

`led_on_color`

`led_rows`

`led_segs`

`led_space`

PtLedBar

PtLedBar — An LED display in bar format.

Synopsis

```
class PtLedBar PtLed
{
}
```

Base Classes

PtWidget <-- PtBasic <-- PtLed <-- PtLedBar

Description

This widget is not yet implemented, and does not appear in the Photon documentation.

PtLedPanel

PtLedPanel — An LED display in panel format.

Synopsis

```
class PtLedPanel PtLed
{
}
```

Base Classes

PtWidget <-- PtBasic <-- PtLed <-- PtLedPanel

Description

This widget is not yet implemented, and does not appear in the Photon documentation.

PtLine

PtLine — A line defined by two points and an origin.

Synopsis

```
class PtLine PtGraphic
{
}
```

Base Classes

PtWidget <-- PtBasic <-- PtGraphic <-- PtLine

Description

This widget is a line defined by a two-element array of start and end points, with respect to a location reference, or origin.

The origin is a PhPoint point, assigned to the widget's inherited origin variable, and the line is positioned relative to it. The start and end points of the line are defined by an array of two PhPoints, assigned to the inherited points variable. Both the origin and the points variables are inherited from the widget's parent, PtGraphic.

For detailed information, please refer to PtLine in the Photon documentation.

Example

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

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

/*
 * This example puts two PtLines with a common origin
 * in a window and prints the origin point and their
 * start and end points.
 */

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

win = new(PtWindow);
win.SetDim (200,120);
win.fill_color = 0xffffbb;

o = new(PhPoint);
o.x = 50;
o.y = 75;

s1 = new(PhPoint);
s1.x = 0;
s1.y = 0;
e1 = new(PhPoint);
e1.x = -15;
e1.y = -25;

s2 = new(PhPoint);
s2.x = 10;
s2.y = 0;
e2 = new(PhPoint);
e2.x = 70;
e2.y = -20;
```

```
ln1 = new(PtLine);
ln1.origin = o;
ln1.points = array(s1,e1);

ln2 = new(PtLine);
ln2.origin = o;
ln2.points = array(s2,e2);

pretty_princ("Line origin: ",ln1.origin,"\n");
pretty_princ("Line1 points: ",ln1.points,"\n");
pretty_princ("Line2 points: ",ln2.points,"\n");

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

PtList

PtList — A scrollable list of items.

Synopsis

```
class PtList PtGenList
{
    items;                // string array  (Pt_ARG_ITEMS)
    list_spacing;         // short  (Pt_ARG_LIST_SPACING)
    selection_indexes;    // unsigned short array  (Pt_ARG_SELECTION_INDEXES)
}
```

Base Classes

PtWidget <-- PtBasic <-- PtContainer <-- PtGenList <-- PtList

Description

This widget is a list of items that can be selected individually or in groups. If the number of items exceeds the size of the widget, a scrollbar is created for accessing them. The widget supports several methods for making single and multiple selections, including browse mode and individual, extended, and range selections. The variables for implementing these options are available from the parent widget, PtGenList.

For detailed information, please refer to PtList in the Photon documentation.

Instance Variables

`items`

An array of strings; each string specifies one list item.

`list_spacing`

A number of pixels specifying the amount of extra spacing between list items. Default is `0`.

`selection_indexes`

An array of index numbers corresponding to the list items that are currently selected.

Callbacks

The following callbacks are associated with this widget:

Callback	Description
Pt_CB_SELECTION	This callback is generated when an item is selected from the list.
Pt_CB_LIST_INPUT	This callback is generated from mouse and key events.

Associated Classes

PtComboBox, PtListCallback, PtListInput, PtCallbackInfo

Example

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

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

/*
 * This example demonstrates a PtList widget.
 */

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

listitems = array();
for (i=0; i<10; i++)
{
  listitems[i] = string("Item number ", i+1);
}

win = new(PtWindow);

lst = new(PtList);
lst.SetDim(150,100);
lst.items = listitems;

PtAttachCallback(lst, Pt_CB_SELECTION,
   'princ("The following indexes have been selected: ",
   (@lst).selection_indexes, "\n"));

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

PtMenu

PtMenu — A pop-up or pull-down menu.

Synopsis

```
class PtMenu PtGroup
{
    menu_flags;         // flag  (Pt_ARG_MENU_FLAGS)
    menu_spacing;       // short  (Pt_ARG_MENU_SPACING)
    menu_text_font;     // string  (Pt_ARG_MENU_TEXT_FONT)
    menu_title;         // string  (Pt_ARG_MENU_TITLE)
    menu_title_font;    // string  (Pt_ARG_MENU_TITLE_FONT)
}
```

Base Classes

PtWidget <-- PtBasic <-- PtContainer <-- PtGroup <-- PtMenu

Description

This widget forms the basis of a menu. Building menus in Gamma is significantly different than in PhAB, however the detailed documentation has not yet been prepared.

For more information, please refer to PtMenu in the Photon documentation.

Instance Variables

menu_flags

This instance variable controls menu behavior, and may be a combination of zero or more of the following flags:

Constant	Description
Pt_MENU_AUTO	Keeps all menu items the same width.
Pt_MENU_TRANSIENT	Destroys the menu when it is closed, possibly even before generating callbacks.
Pt_MENU_TEAR_OFF	Tear Off
Pt_MENU_CHILD	Lets this menu to override its parent.

menu_spacing

A number specifying how many pixels to separate each menu item. The default is the same value as border_width.

menu_text_font

A string specifying the font for the menu text.

menu_title

A string comprising the menu title.

menu_title_font

A string specifying the font for the menu title. Default is "helv12b".

PtMenuBar

PtMenuBar — holds and manages menu buttons.

Synopsis

```
class PtMenuBar PtGroup
{
    menubar_flags;    // flag  (Pt_ARG_MENUBAR_FLAGS)
}
```

Base Classes

PtWidget <-- PtBasic <-- PtContainer <-- PtGroup <-- PtMenuBar

Description

This widget is a menu bar that holds menu buttons. It anchors itself to the top and sides of a window, and automatically aligns its children, which can only be PtMenuButtons.

For detailed information, please refer to PtMenuBar in the Photon documentation.

Instance Variables

menubar_flags

This instance variable controls the widget's behavior, and may be a combination of zero or more of the following flags:

Constant	Description
Pt_MENUBAR_LAST_R_ALIGNED	Last Right Aligned.
Pt_MENUBAR_MENU_OPEN	Internal informational bit.

PtMenuButton

PtMenuButton — A button used to access menus.

Synopsis

```
class PtMenuButton PtContainer
{
    accel_font;     // string  (Pt_ARG_ACCEL_FONT)
    accel_text;     // string  (Pt_ARG_ACCEL_TEXT)
    button_type;    // unsigned short  (Pt_ARG_BUTTON_TYPE)
    offset;         // unsigned short  (Pt_ARG_OFFSET)
}
```

Base Classes

PtWidget <-- PtBasic <-- PtContainer <-- PtMenuButton

Description

This widget is a button located on a menu bar or on a menu itself, which is used to access menus. It functions like a PtButton, but it looks like a PtLabel and is only used on menu bars or menus.

For detailed information, please refer to PtMenuButton in the Photon documentation.

Instance Variables

accel_font

A string specifying the font for the hotkey accelerator text. Default is "helv12".

accel_text

A string specifying the hotkey for this widget.

button_type

This instance variable controls the menu behavior and display, and may have one of the following values:

Constant	Description
Pt_MENU_TEXT	If an 'accelerator' hot key is defined, its text string is displayed.
Pt_MENU_BAR	Bar.
Pt_MENU_RIGHT	The submenu is displayed to the right of the button.
Pt_MENU_DOWN	The submenu is displayed under the button.
Pt_MENU_UP	Up

offset

A number of pixels to offset the accel_text.

PtMenuLabel

PtMenuLabel — provides labels for menus.

Synopsis

```
class PtMenuLabel PtLabel
{
    accel_font;     // string
    accel_text;     // string
    button_type;    // unsigned short
    offset;         // unsigned short
}
```

Base Classes

PtWidget <-- PtBasic <-- PtLabel <-- PtMenuLabel

Description

This widget is not yet documented in detail.

This widget does not appear in the Photon documentation.

Instance Variables

accel_font

A string specifying the font for the hotkey accelerator text. Default is "helv12".

accel_text

A string specifying the hotkey for this widget.

button_type

This instance variable controls the menu behavior and display, and may have one of the following values:

Constant	Description
Pt_MENU_TEXT	If an 'accelerator' hot key is defined, its text string is displayed.
Pt_MENU_BAR	Bar
Pt_MENU_RIGHT	The submenu is displayed to the right of the button.
Pt_MENU_DOWN	The submenu is displayed under the button.
Pt_MENU_UP	Up

offset

A number of pixels to offset the accel_text.

PtMessage

PtMessage — A text message that appears in a pop-up window.

Synopsis

```
class PtMessage PtContainer
{
    msg_button1;    // string  (Pt_ARG_MSG_BUTTON1)
    msg_button2;    // string  (Pt_ARG_MSG_BUTTON2)
    msg_button3;    // string  (Pt_ARG_MSG_BUTTON3)
    msg_default;    // short  (Pt_ARG_MSG_DEFAULT)
    msg_flags;      // flag  (Pt_ARG_MSG_FLAGS)
    msg_font;       // string  (Pt_ARG_MSG_FONT)
    msg_text;       // string  (Pt_ARG_MSG_TEXT)
    msg_title;      // string  (Pt_ARG_MSG_TITLE)
}
```

Base Classes

PtWidget <-- PtBasic <-- PtContainer <-- PtMessage

Description

This widget is a pop-up window with a text message and up to three buttons for user input.

For detailed information, please refer to PtMessage in the Photon documentation.

Instance Variables

msg_button1,
msg_button2,
msg_button3,

A string specifying the message label for each button. The default for msg_button1 is "OK". The other two buttons have a default of nil, and will not be displayed unless this variable is assigned.

msg_default

A number specifying which button is the default choice for the user. The text for this default button will be displayed in bold, and it will be activated if the user presses the **Enter** key.

msg_flags

This instance variable has the following flag:

Constant	Description
Pt_MSG_CENTER_ON_PARENT	Centers the message on its parent.

msg_font

A string specifying the font of the message. Default is "helv12".

msg_text

A string comprising the text of the message.

msg_title

A string comprising a title for the window. Default is nil.

Callbacks

The following callbacks are associated with this widget:

Callback	Description
Pt_CB_MSG_BUTTON1	This callback is generated when button1 is pressed.
Pt_CB_MSG_BUTTON2	This callback is generated when button2 is pressed.
Pt_CB_MSG_BUTTON3	This callback is generated when button3 is pressed.

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();
```

PtMeter

PtMeter — An arc-shaped real-time meter with indicator needle (in QNX 6).

Synopsis

```
class PtMeter PtBasic
{
    meter_color;                // color  (Pt_ARG_METER_COLOR)
    meter_flags;                // flag  (Pt_ARG_METER_FLAGS)
    meter_font_color;           // color  (Pt_ARG_METER_FONT_COLOR)
    meter_increment;            // integer  (Pt_ARG_METER_INCREMENT)
    meter_key_left;             // integer  (Pt_ARG_METER_KEY_LEFT)
    meter_key_right;            // integer  (Pt_ARG_METER_KEY_RIGHT)
    meter_level1_color;         // color  (Pt_ARG_METER_LEVEL1_COLOR)
    meter_level1_pos;           // short  (Pt_ARG_METER_LEVEL1_POS)
    meter_level2_color;         // color  (Pt_ARG_METER_LEVEL2_COLOR)
    meter_level2_pos;           // short  (Pt_ARG_METER_LEVEL2_POS)
    meter_level3_color;         // color  (Pt_ARG_METER_LEVEL3_COLOR)
    meter_max_needle_position;  // short  (Pt_ARG_METER_MAX_NEEDLE_POSITION)
    meter_min_needle_position;  // short  (Pt_ARG_METER_MIN_NEEDLE_POSITION)
    meter_needle_color;         // color  (Pt_ARG_METER_NEEDLE_COLOR)
    meter_needle_position;      // short  (Pt_ARG_METER_NEEDLE_POSITION)
    meter_num_severity_levels;  // short  (Pt_ARG_METER_NUM_SEVERITY_LEVELS)
    meter_text_font;            // string  (Pt_ARG_METER_TEXT_FONT)
}
```

Base Classes

PtWidget <-- PtBasic <-- PtMeter

Description

This widget is a half-circle meter with an indicator needle and up to three different-colored pie-shaped arcs that mark three chosen ranges on the meter. The QNX 4 version of this widget is RtMeter.

For detailed information, please refer to PtMeter in the Photon documentation.

Instance Variables

meter_color

A number specifying the color used for the meter's center, outline, and tick marks. Default is 0x0 (black).

meter_flags

This instance variable controls characteristics of the widget, and may have one or two of the following values:

Constant	Description
PtM_NON_SELECTABLE	The meter is non-interactive.
PtM_SELECTABLE	The meter is interactive. Default is ON.
PtM_NO_TEXT	The minimum and maximum values are not displayed.

`meter_font_color`

A number specifying the color for the display of minimum and maximum values. Default is `0x0` (black).

`meter_increment`

A number of units to move the needle when an assigned key is pressed.

`meter_key_left`
`meter_key_right`

A number or key name from `PkKeyDef.lsp` for the key that moves the needle to the left or right. Defaults are Pk_Left and Pk_Right. You can load the key name constants with a call to **`require_lisp("PkKeyDef.lsp")`**.

`meter_level1_color`

A number specifying a color for the left arc of the meter that corresponds to Level 1. Default is `0x00ff00` (green).

`meter_level1_pos`

A number specifying the end of Level 1, expressed as a percentage of the whole meter. Default is 50. This setting is unaffected by changes to `meter_max_needle_position` or `meter_min_needle_position`.

`meter_level2_color`

A number specifying a color for the center arc of the meter that corresponds to Level 2. Default is `0xffff00` (yellow).

`meter_level2_pos`

A number specifying the end of Level 2, expressed as a percentage of the whole meter. Default is 75. This setting is unaffected by changes to `meter_max_needle_position` or `meter_min_needle_position`.

`meter_level3_color`

A number specifying a color for the right arc of the meter that corresponds to Level 3. Default is `0xff0000` (red).

`meter_max_needle_position`

A number specifying the maximum point on the meter, the maximum value for the needle to register.

`meter_min_needle_position`

A number specifying the minimum point on the meter, the minimum value for the needle to register.

`meter_needle_color`

A number specifying the color of the needle. Default is `0xffffff` (white).

`meter_needle_position`

A number specifying the current position of the needle, within the range of `meter_min_needle_position` and `meter_max_needle_position`. The needle will remain at the maximum or minimum position for any value outside that range.

`meter_num_severity_levels`

The number of arcs that the meter is divided into. The maximum number is 3, which is also the default.

`meter_text_font`

A string specifying the font used for displaying the maximum and minimum values. Default is `"helv10"`.

Associated Classes

`PtMeterCallback`, `PtCallbackInfo`

Example

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

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

/*
 * This example demonstrates an RtMeter in QNX 4, or a
 * PtMeter in QNX 6.
 */

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

function display_value(wnum, wmet)
{
  wmet.meter_needle_position = wnum.numeric_value;
}

function reset_num(wnum, wmet)
{
  wnum.numeric_value = wmet.meter_needle_position;
}

win = new(PtWindow);
win.SetArea(100, 100, 250, 250);
if (_os_ == "QNX4")
     meter = new(RtMeter);
else if (_os_ == "QNX6")
     meter = new(PtMeter);

meter.SetArea(20, 40, 200, 200);
meter.meter_color = 0xccddff;
meter.meter_max_needle_position = 150;
meter.meter_min_needle_position = 0;
meter.meter_level1_color = 0x00ff00;
meter.meter_level1_pos = 50;
meter.meter_level2_color = 0xffff00;
meter.meter_level2_pos = 100;
meter.meter_level3_color = 0xff0000;
meter.meter_needle_position = 70;
meter.meter_needle_color = 0x000000;

num = new(PtNumericInteger);
num.SetArea(100, 150, 50, 20);
num.numeric_increment = 10;
num.numeric_min = 0;
num.numeric_max = 150;
num.numeric_value = 70;
num.numeric_flags = cons(Pt_NUMERIC_WRAP, nil);
num.numeric_flags = cons(Pt_NUMERIC_AUTO_HIGHLIGHT, nil);

label = new(PtLabel);
label.SetPos(10, 190);
label.text_font = "helv09";
label.text_string = string("Adjust the meter by clicking anywhere,\n",
      "by dragging the needle, or by changing \n",
```

```
      "the value in the PtNumeric widget.");

if (_os_ == "QNX4")
     PtAttachCallback(meter, Rt_CB_METER_MOVED, ‘reset_num(@num, @meter));
else if (_os_ == "QNX6")
     PtAttachCallback(meter, Pt_CB_METER_MOVED, ‘reset_num(@num, @meter));

PtAttachCallback(num, Pt_CB_NUMERIC_CHANGED, ‘display_value(@num, @meter));
display_value(num, meter);

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

PtMultiText

PtMultiText — A text box that supports multiple lines and various text styles.

Synopsis

```
class PtMultiText PtContainer
{
    multitext_bottom_line;          // long  (Pt_ARG_MULTITEXT_BOTTOM_LINE)
    multitext_flags;                // flag  (Pt_ARG_MULTITEXT_FLAGS)
    multitext_line_spacing;         // short  (Pt_ARG_MULTITEXT_LINE_SPACING)
    multitext_num_lines;            // integer  (Pt_ARG_MULTITEXT_NUM_LINES)
    multitext_num_lines_visible;    // short  (Pt_ARG_MULTITEXT_NUM_LINES_VISIBLE)
    multitext_rows;                 // long  (Pt_ARG_MULTITEXT_ROWS)
    multitext_top_line;             // long  (Pt_ARG_MULTITEXT_TOP_LINE)
    multitext_wrap_flags;           // flag  (Pt_ARG_MULTITEXT_WRAP_FLAGS)
    multitext_x_scroll_pos;         // short  (Pt_ARG_MULTITEXT_X_SCROLL_POS)
    multitext_y_scroll_pos;         // long  (Pt_ARG_MULTITEXT_Y_SCROLL_POS)
}
```

Base Classes

PtWidget <-- PtBasic <-- PtContainer <-- PtMultiText

Description

This widget is a text box that offers basic word-processing capabilities such as multiple lines, wrapping, cut/copy/paste, range selection, tabs, multiple fonts and so on.

For detailed information, please refer to PtMultiText in the Photon documentation.

Instance Variables

`multitext_bottom_line`

A number. Setting this variable to any value, including `nil`, puts the bottom line of the text at the bottom of the widget display.

`multitext_flags`

This instance variable controls the behavior and display of the widget, and may be a combination of zero or more of the following flags:

Constant	Description
Pt_EMT_AUTOINDENT	Indents a new line to match previous line.
Pt_EMT_FULL_LINES	The line of text requires sufficient space for ascenders and descenders.
Pt_EMT_FORCED_SCROLL	Forces scrolling when a blank space follows text at the end of the widget.
Pt_EMT_SCROLL_TO_CURSOR	Scrolling tracks cursor movements.
Pt_DEFAULT_COLOR	Internal informational bit.
Pt_INHERIT_COLOR	Internal informational bit.
Pt_DEFAULT_FONT	Internal informational bit.
Pt_INHERIT_FONT	Internal informational bit.
Pt_MERGE_PREV	Internal informational bit.
Pt_MERGE_NEXT	Internal informational bit.

`multitext_line_spacing`

A number specifying how many pixels to separate each line of text. Default is `0`.

`multitext_num_lines`

A read-only number that indicates the total number of lines in the text of the widget.

`multitext_num_lines_visible`

A read-only number that indicates the number of lines of text that are visible in the display.

`multitext_rows`

A number of rows that you wish to have visible in the display. This value resizes the widget as necessary, but is applied only once. If the font size changes, the widget won't resize itself automatically; you must assign this variable again to the same value if you wish to keep the same number of rows visible.

`multitext_top_line`

A number indicating which line of text is at the top of the display. The lines are numbered consecutively starting with 1.

`multitext_wrap_flags`

Flags that control how text is wrapped. The default value is `Pt_EMT_WORD` | `Pt_EMT_NEWLINE`, which applies standard word wrapping.

This instance variable may be a combination of zero or more of the following flags:

Constant	Description
`Pt_EMT_WORD`	The text wraps at spaces between words.
`Pt_EMT_CHAR`	The text wraps at the ends of lines.
`Pt_EMT_NEWLINE`	The text wraps at carriage returns.

`multitext_x_scroll_pos`

A number indicating the horizontal scroll position in pixels.

`multitext_y_scroll_pos`

A number indicating which line of text at the top of the display.

Associated Classes

`PtMultiTextAttributes`

Convenience Functions

Arguments

`widget`	A PtMultiText widget.
`start`	The start of the selected text.
`end`	The end of the selected text.
`insert_pos`	The point for insertion of text.
`text`	The source of the text to be inserted.

length	The number of characters to be inserted.
attrs	PtMultiTextAttributes
attributes_mask	

Functions

These functions are extensions of QNX Photon functions. You can refer to the PtMultiText function documentation in QNX Helpviewer for more information about them.

PtMultiTextModifyAttributes (widget, start, end, attrs, attributes_mask) -- applies attributes to the selected range of characters in the *widget*, from *start* to *end*. The attributes are taken from PtMultiTextAttributes, but only the attributes specified in *attributes_mask* are applied.

Returns t.

PtMultiTextModifyText (widget, start, end, insert_pos, text, length, attrs, attributes_mask) -- modifies the contents and attributes of a PtMultiText widget. If *start* and *end* are equal, *length* characters are inserted from *text* at the *insert_pos*. If *start* and *end* are not equal, the selected text is deleted and *length* characters are inserted from *text*.

Returns t.

PtNumeric

PtNumeric — A parent class for numeric widget resources.

Synopsis

```
class PtNumeric PtContainer
{
    numeric_flags;                  // flag  (Pt_ARG_NUMERIC_FLAGS)
    numeric_prefix;                 // string  (Pt_ARG_NUMERIC_PREFIX)
    numeric_suffix;                 // string  (Pt_ARG_NUMERIC_SUFFIX)
    numeric_text_border;            // integer  (Pt_ARG_NUMERIC_TEXT_BORDER)
    numeric_text_bot_border_color;  // color  (Pt_ARG_NUMERIC_TEXT_BOT_BORDER_COLOR)
    numeric_text_color;             // color  (Pt_ARG_NUMERIC_TEXT_COLOR)
    numeric_text_fill_color;        // color  (Pt_ARG_NUMERIC_TEXT_FILL_COLOR)
    numeric_text_font;              // string  (Pt_ARG_NUMERIC_TEXT_FONT)
    numeric_text_top_border_color;  // color  (Pt_ARG_NUMERIC_TEXT_TOP_BORDER_COLOR)
    numeric_updown_border_width;    // integer  (Pt_ARG_NUMERIC_UPDOWN_BORDER_WIDTH)
    numeric_updown_width;           // integer  (Pt_ARG_NUMERIC_UPDOWN_WIDTH)
}
```

Base Classes

PtWidget <-- PtBasic <-- PtContainer <-- PtNumeric

Derived Classes

PtNumericFloat, PtNumericInteger

Description

This class serves as a parent class of resources for numeric widgets, and is not normally instantiated. It joins a PtText widget with a PtUpDown widget, creating a text-entry box for numbers that can be incremented or decremented with small arrow-shaped buttons.

For detailed information, please refer to PtNumeric in the Photon documentation.

Instance Variables

numeric_flags

This instance variable controls the behavior and display of the widget, and may be a combination of zero or more of the following flags:

Constant	Description
Pt_NUMERIC_ENABLE_UPDOWN	Displays the **Up** and **Down** arrow buttons. Default is ON.
Pt_NUMERIC_USE_SEPARATORS	Use comma separators for numbers over 999 (e.g. 5,324,890).
Pt_NUMERIC_WRAP	Wrap numbers from minimum to maximum and from maximum to minimum. Default is ON.
Pt_NUMERIC_AUTO_HIGHLIGHT	Highlight text when selected. Default is ON.

`numeric_prefix`

A string that is prefixed to all numbers entered.

`numeric_suffix`

A string that is appended to all numbers entered.

`numeric_text_border`

A number of pixels specifying the border width of the text display window.

`numeric_text_bot_border_color`

A string specifying a color for the bottom border. Default is `0x606060` (dark grey).

`numeric_text_color`

A string specifying a color for numeric text. Default is `0x0` (black).

`numeric_text_fill_color`

A string specifying a color for the text display. Default is `0xc0c0c0` (grey).

`numeric_text_font`

A string specifying the text font. Default is `"helv12"`.

`numeric_text_top_border_color`

A string specifying a color for the top border. Default is `0xffffff` (white).

`numeric_updown_border_width`

A number of pixels specifying the border width of the `PtUpDown` arrow buttons.

`numeric_updown_width`

A number of pixels specifying the width of the `PtUpDown` arrow buttons.

Callbacks

The following callback is associated with this widget:

Callback	Description
`Pt_CB_NUMERIC_CHANGED`	This callback is generated when the entered value changes.

Associated Classes

`PtNumericFloatCallback`, `PtNumericIntegerCallback`, `PtCallbackInfo`

Example

See the example for `PtNumericInteger`.

PtNumericFloat

PtNumericFloat — An entry box for floating-point values.

Synopsis

```
class PtNumericFloat PtNumeric
{
    numeric_increment;    // double array  (Pt_ARG_NUMERIC_INCREMENT)
    numeric_max;          // double array  (Pt_ARG_NUMERIC_MAX)
    numeric_min;          // double array  (Pt_ARG_NUMERIC_MIN)
    numeric_precision;    // integer  (Pt_ARG_NUMERIC_MAX)
    numeric_value;        // double array  (Pt_ARG_NUMERIC_VALUE)
}
```

Base Classes

PtWidget <-- PtBasic <-- PtContainer <-- PtNumeric <-- PtNumericFloat

Description

This widget is a text-entry box for floating-point numbers that can be incremented or decremented with small arrow-shaped buttons. You can set a minimum, a maximum, a precision level, and the size of the increment/decrement for the arrow buttons. Inherited variables let you use comma separators or add a prefix/suffix strings.

For detailed information, please refer to PtNumericFloat in the Photon documentation.

Instance Variables

numeric_increment

A number specifying the amount to increase or decrease the displayed value when the arrow buttons are pressed. Default is 1.0.

numeric_max

A number specifying the maximum allowable value to be entered. Default is 1,000,000.00.

numeric_min

A number specifying the minimum allowable value to be entered. Default is -1,000,000.00.

numeric_precision

An integer specifying the number of decimal places to display. Default is 2.

numeric_value

The currently displayed value. Default is 0.00.

Callbacks

The following callback is associated with this widget:

Callback	Description
Pt_CB_NUMERIC_CHANGED	This callback is generated when the entered value changes.

Associated Classes

`PtNumericFloatCallback`, `PtCallbackInfo`

Example

See the example for `PtNumericInteger`.

PtNumericInteger

PtNumericInteger — An entry box for integer values.

Synopsis

```
class PtNumericInteger PtNumeric
{
    numeric_increment;    // integer  (Pt_ARG_NUMERIC_INCREMENT)
    numeric_max;          // integer  (Pt_ARG_NUMERIC_MAX)
    numeric_min;          // integer  (Pt_ARG_NUMERIC_MIN)
    numeric_value;        // integer  (Pt_ARG_NUMERIC_VALUE)
}
```

Base Classes

PtWidget <-- PtBasic <-- PtContainer <-- PtNumeric <-- PtNumericInteger

Description

This widget is a text-entry box for integers that can be incremented or decremented with small arrow-shaped buttons. You can set a minimum, a maximum, and the size of the increment/decrement for the arrow buttons. Inherited variables let you use comma separators or add a prefix/suffix strings.

For detailed information, please refer to PtNumericInteger in the Photon documentation.

Instance Variables

numeric_increment

A number specifying the amount to increase or decrease the displayed value when the arrow buttons are pressed. Default is 1.

numeric_max

A number specifying the maximum allowable value to be entered. Default is 1,000,000.

numeric_min

A number specifying the minimum allowable value to be entered. Default is -1,000,000.

numeric_value

The currently displayed value. Default is 0.

Callbacks

The following callback is associated with this widget:

Callback	Description
Pt_CB_NUMERIC_CHANGED	This callback is generated when the entered value changes.

Associated Classes

PtNumericIntegerCallback, PtCallbackInfo

Example

This example, `ex_PtNumeric-RtProgress.g`, is included in the product distribution.

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

/*
 * This example demonstrates PtNumeric and
 * PtProgress widgets.
 */

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

function display_value(wnum, wprog)
{
  wprog.gauge_value = wnum.numeric_value;
}

win = new(PtWindow);
win.SetDim(160, 190);

ilab = new(PtLabel);
ilab.text_string = "Integer values";
ilab.SetArea(20, 6, 120, 20);
ilab.horizontal_alignment = Pt_CENTER;

inum = new(PtNumericInteger);
inum.SetArea(20, 30, 120, 20);
inum.numeric_increment = 5;
inum.numeric_max = 50;
inum.numeric_min = -50;
inum.numeric_value = 5;
inum.numeric_flags = cons(Pt_NUMERIC_WRAP, nil);
inum.numeric_flags = cons(Pt_NUMERIC_AUTO_HIGHLIGHT, nil);

if (_os_ == "QNX4")
     iprog = new(RtProgress);
else
     iprog = new(PtProgress);
iprog.SetArea(20, 60, 115, 20);
iprog.gauge_minimum = -50;
iprog.gauge_maximum = 50;

PtAttachCallback(inum, Pt_CB_NUMERIC_CHANGED, ‘display_value(@inum, @iprog));
display_value(inum, iprog);

if (_os_ == "QNX4")
{
  flab = new(PtLabel);
  flab.text_string = "Float values";
  flab.SetArea(20, 96, 120, 20);
  flab.horizontal_alignment = Pt_CENTER;

  fnum = new(PtNumericFloat);
  fnum.SetArea(20, 120, 120, 20);
  fnum.numeric_increment = 4.8941;
  fnum.numeric_max = 50;
  fnum.numeric_min = -50;
  fnum.numeric_value = 5.34153;
  fnum.numeric_precision = 4;
  fnum.numeric_flags = cons(Pt_NUMERIC_WRAP, nil);
  fnum.numeric_flags = cons(Pt_NUMERIC_AUTO_HIGHLIGHT, nil);

  fprog = new(RtProgress);
  fprog.SetArea(20, 150, 115, 20);
  fprog.gauge_minimum = -50;
  fprog.gauge_maximum = 50;
  fprog.progress_bar_color = 0x0000ff;
```

```
  PtAttachCallback(fnum, Pt_CB_NUMERIC_CHANGED, ‘display_value(@fnum, @fprog));
  display_value(fnum, fprog);
}

else
{
  lab = new(PtLabel);
  lab.SetPos(15, 90);
  lab.text_font  = "helv8";
  lab.text_string = string("Note: This example in QNX 4\n",
      "also contains a PtNumeric Float \n",
      "widget, which is not currently\n",
      "available in Gamma/Photon for\n",
      "QNX 6.");
}

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

PtOnOffButton

PtOnOffButton — A button that can be set to ON or OFF.

Synopsis

```
class PtOnOffButton PtToggleButton
{
    onoff_state;    // short  (Pt_ARG_ONOFF_STATE)
}
```

Base Classes

PtWidget <-- PtBasic <-- PtLabel <-- PtToggleButton <-- PtOnOffButton

Description

This widget is a button that can be set to ON or OFF. It has an indicator box that turns green to indicate an ON state, and grey to indicate an OFF state. A PtOnOffButton can be used inclusively or exclusively in a group with other PtOnOffButtons.

For detailed information, please refer to PtOnOffButton and PtToggleButton in the Photon documentation.

Instance Variables

onoff_state

An integer indicating the state of the button. 0 (the default) is OFF. 1 or any non-zero value is ON.

Callbacks

The following callback is associated with this widget:

Callback	Description
Pt_CB_ONOFF_NEW_VALUE	This callback is generated when the button state (on/off) changes.

Associated Classes

PtOnOffButtonCallback, PtCallbackInfo

PtPane

PtPane — A container that facilitates anchoring and layout of other widgets.

Synopsis

```
class PtPane PtContainer
{
    pane_flags;    // flag
}
```

Base Classes

PtWidget <-- PtBasic <-- PtContainer <-- PtPane

Derived Classes

PtBkgd

Description

This widget defines an area within a window used for anchoring widgets and facilitating layout. Like a window, any part of a child widget that extends beyond the pane's canvas is clipped.

For detailed information, please refer to PtPane in the Photon documentation.

Instance Variables

pane_flags

This instance variable is deprecated. Use the inherited anchor_flags variable instead.

PtPixel

PtPixel — A set of points.

Synopsis

```
class PtPixel PtGraphic
{
}
```

Base Classes

PtWidget <-- PtBasic <-- PtGraphic <-- PtPixel

Description

This widget is a group of points, located with respect to an origin point. The origin is a PhPoint, assigned to the origin variable (inherited from PtGraphic). The group of points is defined by an array of PhPoints assigned to the points variable (also inherited from PtGraphic).

For detailed information, please refer to PtPixel in the Photon documentation.

Example

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

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

/*
 * This example puts a PtPixel widget with three points
 * into a window, and prints its array of points.
 */

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

win = new(PtWindow);
win.SetDim (100,100);
win.fill_color = 0xffffbb;

p1 = new(PhPoint);
p1.x = 5;
p1.y = 5;

p2 = new(PhPoint);
p2.x = 8;
p2.y = 5;

p3 = new(PhPoint);
p3.x = 5;
p3.y = 8;

o = new(PhPoint);
o.x = 45;
o.y = 45;

pix = new(PtPixel);
pix.points = array(p1,p2,p3);
pix.origin = o;

pretty_princ("PtPixel points:\n",pix.points,"\n");

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

PtPolygon

PtPolygon — A set of points connected by lines.

Synopsis

```
class PtPolygon PtGraphic
{
    polygon_flags;    // flag  (Pt_ARG_POLYGON_FLAGS)
}
```

Base Classes

PtWidget <-- PtBasic <-- PtGraphic <-- PtPolygon

Description

This widget is a polygon defined by a set of points that act as vertices, located with respect to an origin point. The origin is a PhPoint, assigned to the origin variable (inherited from PtGraphic). The set of points is defined by an array of PhPoints assigned to the points variable (also inherited from PtGraphic).

The polygon can be open or closed, empty or filled, and drawn using absolute or relative coordinates. Its lines can be stroked or unstroked.

For detailed information, please refer to PtPolygon in the Photon documentation.

Instance Variables

polygon_flags

This instance variable determines the characteristics of the polygon, and may be a combination of zero or more of the following flags:

Constant	Description
Pg_CLOSED	The end point is connected to the start point.
Pg_POLY_STROKE	The polygon is outlined.
Pg_POLY_RELATIVE	Use relative coordinates for drawing.
Pg_POLY_FILL	The polygon is filled.

Example

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

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

/*
 * This example puts a PtPolygon into a window,
 * and prints its array of vertex points.
 */

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

win = new(PtWindow);
win.SetDim (150,150);
```

```
win.fill_color = 0xffffbb;

p1 = new(PhPoint);
p1.x = 5;
p1.y = 5;

p2 = new(PhPoint);
p2.x = 60;
p2.y = -30;

p3 = new(PhPoint);
p3.x = 80;
p3.y = 40;

o = new(PhPoint);
o.x = 20;
o.y = 65;

poly = new(PtPolygon);
poly.points = array(p1,p2,p3);
poly.origin = o;
poly.polygon_flags = Pg_POLY_FILL;
poly.fill_color = 0x00ddbb;

pretty_princ("PtPolygon points:\n",poly.points,"\n");

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

PtRaw

PtRaw — A canvas for drawing Photon graphics primitives.

Synopsis

```
class PtRaw PtBasic
{
    raw_connect_f;    // unimplemented
    raw_draw_f;       // unimplemented
    raw_extent_f;     // unimplemented
    raw_init_f;       // unimplemented
}
```

Base Classes

PtWidget <-- PtBasic <-- PtRaw

Description

This widget is a canvas for drawing custom widgets. Its instance variables are not yet implemented in Gamma.

For more information, please refer to PtRaw in the Photon documentation.

PtRect

PtRect — A rectangle defined by two points.

Synopsis

```
class PtRect PtGraphic
{
    rect_roundness;    // unsigned short  (Pt_ARG_RECT_ROUNDNESS)
}
```

Base Classes

PtWidget <-- PtBasic <-- PtGraphic <-- PtRect

Description

This widget is a rectangle defined by a two points that are the upper-left and lower-right hand corners. It is located with respect to an origin point.

The origin is a PhPoint, assigned to the origin variable (inherited from PtGraphic). The two points are defined by an array of PhPoints assigned to the points variable (also inherited from PtGraphic).

For detailed information, please refer to PtRect in the Photon documentation.

Instance Variables

rect_roundness

A number of pixels specifying the radius for the corners of the rectangle. Default is 0 (not rounded at all).

Example

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

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

/*
 * This example puts a PtRect widget into a window,
 * and prints its two corner points.
 */

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

win = new(PtWindow);
win.SetDim (150,150);
win.fill_color = 0xffffbb;

p1 = new(PhPoint);
p1.x = 0;
p1.y = 0;

p2 = new(PhPoint);
p2.x = 90;
p2.y = 50;

o = new(PhPoint);
o.x = 25;
o.y = 25;
```

```
rect = new(PtRect);
rect.points = array(p1,p2);
rect.origin = o;
rect.rect_roundness = 10;

pretty_princ("PtRect points:\n",rect.points,"\n");

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

PtRegion

PtRegion — A means to access and control regions of the interface.

Synopsis

```
class PtRegion PtContainer
{
    region_fields;          // flag  (Pt_ARG_REGION_FIELDS)
    region_flags;           // flag  (Pt_ARG_REGION_FLAGS)
    region_handle;          // unsigned long  (Pt_ARG_REGION_HANDLE)
    region_infront;         // region ID  (Pt_ARG_REGION_INFRONT)
    region_input_group;     // unsigned short  (Pt_ARG_REGION_INPUT_GROUP)
    region_opaque;          // flag  (Pt_ARG_REGION_OPAQUE)
    region_owner;           // long  (Pt_ARG_REGION_OWNER)
    region_parent;          // region ID  (Pt_ARG_REGION_PARENT)
    region_sense;           // flag  (Pt_ARG_REGION_SENSE)
}
```

Base Classes

PtWidget <-- PtBasic <-- PtContainer <-- PtRegion

Description

This is a widget representation of the PhRegion class, providing access and control of a region in the widget environment.

For detailed information, please refer to PtRegion in the Photon documentation.

Instance Variables

region_fields

Most of these flags control access to instance variables. They must be set before any changes for that variable will take effect.

This instance variable may be a combination of zero or more of the following flags:

Constant	Description
Ph_REGION_BEHIND	Specifies the region behind this region when it is opened. 0 puts this region behind all brothers.
Ph_REGION_DATA	Reserved for future implementation.
Ph_REGION_EV_OPAQUE	Enables the region_opaque variable.
Ph_REGION_EV_SENSE	Enables the region_sense variable.
Ph_REGION_FLAGS	Enables the region_flags variable.
Ph_REGION_HANDLE	Enables the region_handle variable.
Ph_REGION_IN_FRONT	Enables the region_in_front variable.
Ph_REGION_INPUT_GROUP	Enables the region_input_group variable.
Ph_REGION_ORIGIN	Registers the region's origin point data.
Ph_REGION_OWNER	Enables the region_owner variable.
Ph_REGION_PARENT	Enables the region_parent variable.
Ph_REGION_RECT	Registers the region's dimension and position data.

Constant	Description
Ph_REGION_STATE	Is read only--not to be modified.

region_flags

This instance variable specifies region characteristics, and may be a combination of zero or more of the following flags:

Constant	Description
Ph_WINDOW_REGION	Defines the region as a window.
Ph_WND_MGR_REGION	Defines the region as a window manager.
Ph_GRAFX_REGION	Defines the region as a graphics region.
Ph_PTR_REGION	Defines the region as a pointer region.
Ph_KBD_REGION	Defines the region as a keyboard.
Ph_PRINT_REGION	Defines the region as a print region.
Ph_INPUTGROUP_REGION	Defines the region as a input group.
Ph_AUXPTR_REGION	Defines the region as a auxiliary pointer region.
Ph_FORCE_FRONT	Forces the region to the front.
Ph_FOLLOW_IG_SIZE	Makes the region follow the input group size.
Ph_FORCE_BOUNDARY	Forces a boundary on the region.
Ph_NO_COMPRESSION	Prevents compression of the region.
Ph_CURSOR_SET	Cursor Set

region_handle

A widget pointer that specifies the widget associated with the region. Set to 0 or leave the default. This variable's corresponding flag must be set in the region_fields variable.

region_infront

A region ID number that specifies the region in front of this region when it is opened. 0 puts this region in front of all brothers. This variable's corresponding flag must be set in the region_fields variable.

region_input_group

A number that specifies the region's input group. This variable's corresponding flag must be set in the region_fields variable.

region_opaque

Flags that specify any events that the region is opaque to. This variable's corresponding flag must be set in the region_fields variable.

region_owner

A number that specifies the owner of the region. This variable's corresponding flag must be set in the region_fields variable.

region_parent

A region ID number that specifies the parent region for this region. This variable's corresponding flag must be set in the region_fields variable.

`region_sense`

Flags that specify events the region is sensitive to. This variable's corresponding flag must be set in the `region_fields` variable.

PtScale

PtScale — A marked scale.

Synopsis

```
class PtScale PtGauge
{
    scale_flags;                // flag
    scale_font;                 // string
    scale_major_tick_color;     // color
    scale_major_tick_division;  // short
    scale_major_tick_length;    // short
    scale_minor_tick_color;     // color
    scale_minor_tick_division;  // short
    scale_minor_tick_length;    // short
}
```

Base Classes

PtWidget <-- PtBasic <-- PtGauge <-- PtScale

Description

This widget is a horizontal or vertical scale that features major and minor ticks.

This widget does not appear in the Photon documentation.

Instance Variables

scale_flags

This instance variable specifies the general features of the scale, and may be a combination of zero or more of the following flags:

Constant	Description
Pt_SCALE_HORIZONTAL	The scale is oriented horizontally.
Pt_SCALE_VERTICAL	The scale is oriented vertically.
Pt_SCALE_REVERSE	The max and min values of the scale are reversed.
Pt_SCALE_DRAW_TOP_LINE	Draw Top Line
Pt_SCALE_LABEL	A label is associated with the scale.

scale_font

A string specifying the font of the label.

scale_major_tick_color

A number specifying the color of the major tick marks. Default is 0x0 (black).

scale_major_tick_division

An integer specifying into how many equal-sized major parts the scale should be divided into. Default is 5.

scale_major_tick_length

A number of pixels specifying the length of the major tick marks. Default is 10.

`scale_minor_tick_color`

A number specifying the color of the minor tick marks. Default is `0x0` (black).

`scale_minor_tick_division`

An integer specifying into how many equal-sized minor parts each major division should be divided into. Default is `3`.

`scale_minor_tick_length`

A number of pixels specifying the length of the minor tick marks. Default is `5`.

Example

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

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

/*
 * This example demonstrates two PtScales, one using
 * user_defined values for instance variables, the other
 * with default values.  The PtScale widget is only
 * available in QNX 4.
 */

if (_os_ == "QNX6")
     princ ("\nThis widget is not available for QNX 6.", "\n\n");
else
{
  require_lisp("PhotonWidgets.lsp");
  PtInit(nil);

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

  scl = new(PtScale);
  scl.SetArea(20,25,50,250);
  scl.fill_color = 0xffffaa;
  scl.scale_flags = Pt_SCALE_VERTICAL;
  scl.scale_font = "helv12";
  scl.scale_major_tick_color = 0xff0000;
  scl.scale_major_tick_division = 10;
  scl.scale_major_tick_length = 15;
  scl.scale_minor_tick_color = 0x0000ff;
  scl.scale_minor_tick_division = 5;
  scl.scale_minor_tick_length = 6;

  scldeflt = new(PtScale);
  scldeflt.SetArea(80,25,200,20);

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

PtScrollArea

PtScrollArea — A compact display of a large virtual area.

Synopsis

```
class PtScrollArea PtContainer
{
    scroll_area_flags;          // flag  (Pt_ARG_SCROLL_AREA_FLAGS)
    scroll_area_increment_x;    // unsigned short  (Pt_ARG_SCROLL_AREA_INCREMENT_X)
    scroll_area_increment_y;    // unsigned short  (Pt_ARG_SCROLL_AREA_INCREMENT_Y)
    scroll_area_max_x;          // unsigned short  (Pt_ARG_SCROLL_AREA_MAX_X)
    scroll_area_max_y;          // unsigned short  (Pt_ARG_SCROLL_AREA_MAX_Y)
    scroll_area_pos_x;          // unsigned short  (Pt_ARG_SCROLL_AREA_POS_X)
    scroll_area_pos_y;          // unsigned short  (Pt_ARG_SCROLL_AREA_POS_Y)
    scrollbar_x_display;        // unsigned short  (Pt_ARG_SCROLLBAR_X_DISPLAY)
    scrollbar_x_height;         // unsigned short  (Pt_ARG_SCROLLBAR_X_HEIGHT)
    scrollbar_y_display;        // unsigned short  (Pt_ARG_SCROLLBAR_Y_DISPLAY)
    scrollbar_y_width;          // unsigned short  (Pt_ARG_SCROLLBAR_Y_WIDTH)
}
```

Base Classes

PtWidget <-- PtBasic <-- PtContainer <-- PtScrollArea

Description

This widget is physical window that can display part of what is usually a larger virtual area. Scrollbars are provided for moving non-visible portions of the virtual area into the display area.

For detailed information, please refer to PtScrollArea in the Photon documentation.

Instance Variables

scroll_area_flags

This instance variable controls scrolling behavior, and may be a combination of zero or more of the following flags:

Constant	Description
Pt_SCROLL_AREA_IGNORE_KEYS	Makes the scroll area ignore the **PgUp**, **PgDn**, **Home**, **End** and arrow keys.
Pt_SCROLL_AREA_TRACK_FOCUS	Enables auto-scrolling to keep widgets visible when being moved.

scroll_area_increment_x
scroll_area_increment_y

A number specifying the number of pixels to scroll the display in the x or y direction when the scrollbar arrow is clicked. Default is 10.

`scroll_area_max_x`
`scroll_area_max_y`

A number specifying the maximum width (x) or height (y) of the widget's virtual display. The visible portion of this area can be specified with the inherited `area` variable.

`scroll_area_pos_x`
`scroll_area_pos_y`

A number of pixels specifying the horizontal (x) or vertical (y) position of the virtual area.

`scrollbar_x_display`
`scrollbar_y_display`

These instance variables control the whether horizontal (x) or vertical (y) scrollbars appear. Each variable may have one of the following values:

Constant	Description
`Pt_NEVER`	Never show the scrollbar.
`Pt_ALWAYS`	Always show the scrollbar.
`Pt_AS_REQUIRED`	Show the scrollbar when the virtual area is greater than the display area.

`scrollbar_x_height`
`scrollbar_y_width`

A number of pixels specifying the scrollbar trough size, which means the trough height for horizontal scrollbars and trough width for vertical scrollbars. The minimum is 6. The default value, 0, sets the size to 15.

Callbacks

The following callbacks are associated with this widget:

Callback	Description
`Pt_CB_SCROLLED_X`	This callback is generated when a `PtScrollArea` widget's horizontal scrollbar is moved.
`Pt_CB_SCROLLED_Y`	This callback is generated when a `PtScrollArea` widget's vertical scrollbar is moved.

PtScrollbar

PtScrollbar — A sliding handle for navigating a scroll area.

Synopsis

```
class PtScrollbar PtBasic
{
    increment;          // long  (Pt_ARG_INCREMENT)
    maximum;            // integer  (Pt_ARG_MAXIMUM)
    min_slider_size;    // integer  (Pt_ARG_MIN_SLIDER_SIZE)
    minimum;            // integer  (Pt_ARG_MINIMUM)
    orientation;        // flags  (Pt_ARG_ORIENTATION)
    page_increment;     // integer  (Pt_ARG_PAGE_INCREMENT)
    scroll_position;    // integer  (Pt_ARG_SCROLL_POSITION)
    scrollbar_flags;    // flag  (Pt_ARG_SCROLLBAR_FLAGS)
    slider_size;        // integer  (Pt_ARG_SLIDER_SIZE)
}
```

Base Classes

PtWidget <-- PtBasic <-- PtScrollbar

Description

This widget is a vertical or horizontal scrollbar with a handle that slides in a trough to set and show the position of the scroll area over the virtual area of a PtScrollArea widget. The scrollbar has arrow buttons at either end to move the handle.

The scrollbar trough represents a range of values, which is divided into integer increments. These increments are the unit of measure for the size, position, and movement of the handle. The number of increments is determined by the difference between the maximum and minimum variables.

For additional information, please refer to PtScrollbar in the Photon documentation.

Instance Variables

increment

A number specifying the number of increments to move the handle when an arrow button is pressed. Default is 1.

maximum

An integer specifying the highest increment number, at the end of the scrollbar range. Default is 19. The total number of increments is calculated by subtracting the minimum value from this value.

min_slider_size

An integer specifying the minimum size (in pixels) that the handle is allowed to become. Default is 5 pixels.

minimum

An integer specifying the lowest increment number, at the beginning of the scrollbar range. Default is 0. The total number of increments is calculated by subtracting this value from the maximum value.

orientation

This instance variable determines the horizontal/vertical orientation of the scrollbar, and may have one of the following values:

Constant	Description
Pt_HORIZONTAL	The scrollbar is oriented horizontally.
Pt_VERTICAL	The scrollbar is oriented vertically. This is the default.

page_increment

An integer that specifies the number of increments the handle moves when the trough is clicked. Default is -1, which causes the handle to move the distance of its own length.

scroll_position

An integer specifying the position of the top of the handle in the range.

scrollbar_flags

This instance variable specifies several scrollbar characteristics, and may be a combination of zero or more of the following flags:

Constant	Description
Pt_SCROLLBAR_HORIZONTAL	Change the scrollbar from vertical to horizontal.
Pt_SCROLLBAR_SHOW_ARROWS	Show arrow buttons at both ends of the scrollbar.
Pt_SCROLLBAR_INVERTED	Invert the scrolling mechanism: max/min or left/right.
Pt_SCROLLBAR_FOCUSED	Render the scrollbar as if it had focus, even when it doesn't.

slider_size

An integer specifying the number of increments to size the handle. Default is 20.

Callbacks

The following callbacks are associated with this widget:

Callback	Description
Pt_CB_SCROLL_MOVE	This callback is generated when the scroll position changes.

Associated Classes

PtScrollbarCallback, PtCallbackInfo

PtSeparator

PtSeparator — A line that separates menu items.

Synopsis

```
class PtSeparator PtBasic
{
    sep_flags;    // flag  (Pt_ARG_SEP_FLAGS)
    sep_type;     // unsigned short  (Pt_ARG_SEP_TYPE)
}
```

Base Classes

PtWidget <-- PtBasic <-- PtSeparator

Description

This widget is a separator line for menu items. It has several styles.

For detailed information, please refer to PtSeparator in the Photon documentation.

Instance Variables

sep_flags

This instance variable determines the orientation of the widget.

Constant	Description
Pt_SEP_ORIENTATION	Set to the value of Pt_SEP_VERTICAL or Pt_SEP_HORIZONTAL.
Pt_SEP_VERTICAL	Makes the separator vertical.
Pt_SEP_HORIZONTAL	Makes the separator horizontal.

sep_type

This instance variable specifies the type of line, and may have one of the following values:

Constant	Description
Pt_SINGLE_LINE	Single Line
Pt_DOUBLE_LINE	Double Line
Pt_SINGLE_DASH_LINE	Single Dash Line
Pt_DOUBLE_DASH_LINE	Double Dash Line
Pt_ETCHED_IN	Etched In
Pt_ETCHED_OUT	Etched Out
Pt_NOLINE	No Line

PtSlider

PtSlider — A sliding-scale input mechanism.

Synopsis

```
class PtSlider PtGauge
{
    slider_flags;             // flag  (Pt_ARG_SLIDER_FLAGS)
    slider_handle_height;     // unsigned short  (Pt_ARG_SLIDER_HANDLE_HEIGHT)
    slider_handle_width;      // unsigned short  (Pt_ARG_SLIDER_HANDLE_WIDTH)
    slider_image;             // PhImage  (Pt_ARG_SLIDER_IMAGE)
    slider_increment;         // unsigned short  (Pt_ARG_SLIDER_INCREMENT)
    slider_label_br;          // string  (Pt_ARG_SLIDER_LABEL_BR)
    slider_label_br_col;      // color  (Pt_ARG_SLIDER_LABEL_BR_COL)
    slider_label_tl;          // string  (Pt_ARG_SLIDER_LABEL_TL)
    slider_label_tl_col;      // color  (Pt_ARG_SLIDER_LABEL_TL_COL)
    slider_multiple;          // unsigned short  (Pt_ARG_SLIDER_MULTIPLE)
    slider_tick_major_col;    // color  (Pt_ARG_SLIDER_TICK_MAJOR_COL)
    slider_tick_major_div;    // long  (Pt_ARG_SLIDER_TICK_MAJOR_DIV)
    slider_tick_major_len;    // unsigned short  (Pt_ARG_SLIDER_TICK_MAJOR_LEN)
    slider_tick_minor_col;    // color  (Pt_ARG_SLIDER_TICK_MINOR_COL)
    slider_tick_minor_div;    // long  (Pt_ARG_SLIDER_TICK_MINOR_DIV)
    slider_tick_minor_len;    // unsigned short  (Pt_ARG_SLIDER_TICK_MINOR_LEN)
    slider_trough_col;        // color  (Pt_ARG_SLIDER_TROUGH_COL)
    slider_trough_size;       // unsigned short  (Pt_ARG_SLIDER_TROUGH_SIZE)
}
```

Base Classes

PtWidget <-- PtBasic <-- PtGauge <-- PtSlider

Description

This widget is a sliding scale that accepts numerical data input in analog fashion. It consists of a handle that moves along a trough and points to optional tick marks.

For detailed information, please refer to PtSlider in the Photon documentation.

Instance Variables

slider_flags

This instance variable specifies the location of the tick marks, and may be a combination of zero or more of the following flags:

Constant	Description
Pt_TICKS_ON_TOP	Put the scale marks on top of the trough.
Pt_TICKS_ON_LEFT	Put the scale marks to the left of the trough.
Pt_TICKS_ON_BOTTOM	Put the scale marks on the bottom of the trough.
Pt_TICKS_ON_RIGHT	Put the scale marks to the right of the trough.
Pt_TICKS_TOUCH_TROUGH	Make the scale marks touch the trough.
Pt_TICKS_ETCHED_IN	Give the scale marks an etched-in appearance.
Pt_TICKS_ETCHED_OUT	Give the scale marks an etched-out appearance.
Pt_SLIDER_POINT_LEFT	The handle will point left.
Pt_SLIDER_POINT_UP	The handle will point up.
Pt_SLIDER_POINT_RIGHT	The handle will point right.

Constant	Description
Pt_SLIDER_POINT_DOWN	The handle will point down.
Pt_SLIDER_IMAGE	The slider handle is an image specified by slider_image..
Pt_SLIDER_MASK	Internal informational bit.

slider_handle_height
slider_handle_width

A number of pixels specifying the height or width of the handle. Defaults are 40 and 15 respectively.

slider_image

A PhImage to display on the handle. You must set the slider_flags Pt_SLIDER_IMAGE in order to use this variable.

slider_increment

An integer specifying the increment for the slider to move when the arrow keys are pressed. Default is 1.

slider_label_br

A string comprising the label to display on the bottom or right of the slider, depending on its orientation.

slider_label_br_col

A number specifying the color of the bottom or right label. Default is 0x0 (black).

slider_label_tl

A string comprising the label to display on the top or left of the slider, depending on its orientation.

slider_label_tl_col

A number specifying the color of the top or left label. Default is 0x0 (black).

slider_multiple

An integer specifying the increment for the slider to move when you press **PgUp** or **PgDn**, or click on the trough. Default is 0.

slider_tick_major_col
slider_tick_minor_col

A number specifying a color for the major or minor ticks. Default is 0x0 (black).

slider_tick_major_div

The number of major tick divisions. Default is 10.

slider_tick_major_len
slider_tick_minor_len

A number of pixels specifying the length of the major or minor ticks. Defaults are 10 and 0 respectively.

slider_tick_minor_div

The number of minor tick divisions per major division. Default is 0.

slider_trough_col

A number specifying the color of the trough. Default is 0xc0c0c0 (grey).

`slider_trough_size`

A number of pixels specifying the width of the trough. Default is `4`.

Callbacks

The following callback is associated with this widget:

Callback	Description
`Pt_CB_SLIDER_MOVE`	This callback is generated when the position of the slider handle changes.

Associated Classes

`PtSliderCallback`, `PtCallbackInfo`

PtTerminal

PtTerminal — A window that functions like a terminal.

Synopsis

```
class PtTerminal PtContainer
{
    term_color_table;     // color array  (Pt_ARG_TERM_COLOR_TABLE)
    term_cols;            // unsigned short  (Pt_ARG_TERM_COLS)
    term_console;         // console string  (Pt_ARG_TERM_CONSOLE)
    term_cur_col;         // short  (Pt_ARG_TERM_CUR_COL)
    term_cur_pos;         // PhPoint  (Pt_ARG_TERM_CUR_POS)
    term_cur_row;         // short  (Pt_ARG_TERM_CUR_ROW)
    term_cursor_flags;    // flag  (Pt_ARG_TERM_CURSOR_FLAGS)
    term_draw_modes;      // unsigned char  (Pt_ARG_TERM_DRAW_MODES)
    term_font;            // string  (Pt_ARG_TERM_FONT)
    term_font_index;      // short  (Pt_ARG_TERM_FONT_INDEX)
    term_font_list;       // string array  (Pt_ARG_TERM_FONT_LIST)
    term_font_size;       // PhDim  (Pt_ARG_TERM_FONT_SIZE)
    term_margins;         // PhRect  (Pt_ARG_TERM_MARGINS)
    term_maxcols;         // short  (Pt_ARG_TERM_MAXCOLS)
    term_maxrows;         // short  (Pt_ARG_TERM_MAXROWS)
    term_maxsize;         // PhPoint  (Pt_ARG_TERM_MAXSIZE)
    term_mincols;         // short  (Pt_ARG_TERM_MINCOLS)
    term_minrows;         // short  (Pt_ARG_TERM_MINROWS)
    term_minsize;         // PhPoint  (Pt_ARG_TERM_MINSIZE)
    term_options;         // flag  (Pt_ARG_TERM_OPTIONS)
    term_optmask;         // flag  (Pt_ARG_TERM_OPTMASK)
    term_protocol;        // short  (Pt_ARG_TERM_PROTOCOL)
    term_resize_fl;       // flag  (Pt_ARG_TERM_RESIZE_FL)
    term_resize_str;      // string  (Pt_ARG_TERM_RESIZE_STR)
    term_rows;            // unsigned short  (Pt_ARG_TERM_ROWS)
    term_scrlbk_count;    // unsigned short  (Pt_ARG_TERM_SCRLBK_COUNT)
    term_scrlbk_limit;    // short  (Pt_ARG_TERM_SCRLBK_LIMIT)
    term_scrlbk_pos;      // short  (Pt_ARG_TERM_SCRLBK_POS)
    term_scroll;          // unsigned short  (Pt_ARG_TERM_SCROLL)
    term_size;            // PhPoint  (Pt_ARG_TERM_SIZE)
    term_visual_bell;     // short  (Pt_ARG_TERM_VISUAL_BELL)
}
```

Base Classes

PtWidget <-- PtBasic <-- PtContainer <-- PtTerminal

Derived Classes

PtTty

Description

This widget is a terminal window that can be accessed within your application.

For detailed information, please refer to PtTerminal in the Photon documentation.

Instance Variables

term_color_table

An array of colors used by the terminal. Default is the 16 standard CGA colors.

`term_cols`

A number specifying the number of columns of characters displayed on the terminal. Default is `80`.

`term_console`

Provides for console emulation. Details not yet documented.

`term_cur_col`

A number specifying the cursor's column position.

`term_cur_pos`

A number specifying the cursor's row and column position.

`term_cur_row`

A number specifying the cursor's row position.

`term_cursor_flags`

This instance variable controls various cursor characteristics, and may be a combination of zero or more of the following flags:

Constant	Description
`Pt_TERM_CURSOR_NEVER`	Block cursor blinking.
`Pt_TERM_CURSOR_ON_FOCUS`	Enable cursor blinking when the widget has focus.
`Pt_TERM_CURSOR_ALWAYS`	Enable cursor blinking always.
`Pt_TERM_CURSOR_TIMER`	Activate the cursor timer, whether needed or not.
`Pt_TERM_CURSOR_NOSPEEDCHK`	Keeps the cursor blinking even if the Photon connection is slow.

`term_draw_modes`

This instance variable controls blitting and redrawing of the terminal, and may be a combination of zero or more of the following flags:

Constant	Description
`Pt_TERM_SCROLL_NOBLIT`	Always redraw--don't blit.
`Pt_TERM_SCROLL_NOHWCHK`	Don't check for hardware support before attempting to blit.
`Pt_TERM_SCROLL_RFSH`	Refresh the screen upon scrolling, even if PtBlit() isn't called.
`Pt_TERM_SCROLL_NOVISCHK`	Assume widget is totally visible--not clipped or obscured.
`Pt_TERM_SCROLL_NOSPEEDCHK`	Don't check bandwidth.

`term_font`

A string specifying the font used by the terminal. Default is `"pcterm14"`. Only fixed-width fonts are accepted.

`term_font_index`

An index number into the `term_font_list` of the font currently in use. The default, -1, is used if a requested font is not on the list.

`term_font_list`

An array of strings, each of which is the name of a font.

`term_font_size`

A read-only number indicating the size of the font.

`term_margins`

A `PhRect` class whose lower right and upper left point coordinate values specify the number of pixels in the lower, right, upper, and left margins.

`term_maxcols`

A number specifying the maximum number of columns allowed for the terminal. Default is `1000`.

`term_maxrows`

A number specifying the maximum number of rows allowed for the terminal. Default is `1000`.

`term_maxsize`

A `PhPoint` whose x and y coordinate values specify the maximum number of columns and rows allowed for the terminal. Default is:

```
{PhPoint (x . 1000) (y . 1000)}
```

`term_mincols`

A number specifying the minimum number of columns allowed for the terminal. Default is `1`.

`term_minrows`

A number specifying the minimum number of rows allowed for the terminal. Default is `1`.

`term_minsize`

A `PhPoint` whose x and y coordinate values specify the minimum number of columns and rows allowed for the terminal. Default is:

```
{PhPoint (x . 1) (y . 1)}
```

`term_options`

Provides for various terminal options. Details not yet documented.

`term_optmask`

A mask for `term_options`. Details not yet documented.

`term_protocol`

A number specifying the protocol. 0 specifies QNX 4, 1 (the default) specifies ANSI.

`term_resize_fl`

This instance variable controls resizing of the terminal, and may be a combination of zero or more of the following flags:

Constant	Description
`Pt_TERM_ANCHOR_PARENT_WIDTH`	If the terminal's right edge is anchored to the parent right edge, the parent width is resized.
`Pt_TERM_ANCHOR_PARENT_HEIGHT`	If the terminal's bottom is anchored to the parent bottom, the parent height is resized.
`Pt_TERM_ANCHOR_WINDOWS_ONLY`	The parent is not resized unless it is a window.
`Pt_TERM_OPFONT`	Can use escape sequences to set the font.
`Pt_TERM_KBFONT`	Can use the keyboard to set the font.
`Pt_TERM_KBFORCE`	The widget size or dim stays constant when font sizes are changed with the keyboard.

`term_resize_str`

A string specifying resize parameters. Details not yet documented.

`term_rows`

A number specifying the number of rows displayed on the terminal. Default is `25`.

`term_scrlbk_count`

A number indicating the current number of lines in the scrollback buffer.

`term_scrlbk_limit`

A number specifying the maximum number of lines that can be saved in the scrollback buffer.

`term_scrlbk_pos`

A number specifying the current position in the scrollback buffer. Any output resets this number to the default, 0.

`term_scroll`

A number specifying a value used to optimize scrolling. Details are not yet documented.

`term_size`

A PhPoint specifying the number of rows and columns that make up the screen size. Default is:

```
{PhPoint (x . 25) (y . 80)}
```

`term_visual_bell`

A number of milliseconds that the screen will flash when the ASCII BEL character (**Ctrl-G**) is pressed.

Callbacks

The following callbacks are associated with this widget:

Callback	Description
`Pt_CB_TERM_INPUT`	This callback is generated when mouse or keyboard input is received.

Callback	Description
`Pt_CB_TERM_FONT`	This callback is generated when the font changes.
`Pt_CB_TERM_RESIZE`	This callback is generated when the terminal is going to change size (rows/columns).
`Pt_CB_TERM_RESIZED`	This callback is generated when the terminal size has changed.
`Pt_CB_TERM_OPTIONS`	This callback is generated when the term_options resource value changes.
`Pt_CB_TERM_APP`	Application
`Pt_CB_TERM_SCRLBK`	This callback is generated when the term_scrlbk_pos changes.

Associated Classes

`PtCallbackInfo`, `PtTerminalFontChange`, `PtTerminalInput`, `PtTerminalOptionChange`, `PtTerminalScrlbkCb`, `PtTerminalSizeChange`

Convenience Functions

Arguments

`widget`	`PtTerminal`
`data`	
`length`	

Function

This function is an extension of the QNX Photon function. You can refer to the PtTerminal function documentation in QNX Helpviewer for more information about it.

`PtTerminalPut` `(widget, data, length?)` -- outputs characters to the terminal widget. If an error occurs, the `errno` is set.

Returns 0 on completion, or -1 for error.

PtText

PtText — A single-line or multi-line text entry/display field.

Synopsis

```
class PtText PtLabel
{
    columns;            // short  (Pt_ARG_COLUMNS)
    cursor_position;    // short  (Pt_ARG_CURSOR_POSITION)
    edit_mask;          // string  (Pt_ARG_EDIT_MASK)
    max_length;         // short  (Pt_ARG_MAX_LENGTH)
    text_flags;         // flag  (Pt_ARG_TEXT_FLAGS)
}
```

Base Classes

PtWidget <-- PtBasic <-- PtLabel <-- PtText

Description

This widget is a box that accepts text entry and displays text strings. Its cursor toggles between insert and replace modes, and can be used to select ranges of text. Possible callbacks include text changes, cursor movements, and pressing the **Enter** key.

For detailed information, please refer to PtText in the Photon documentation.

Instance Variables

`columns`

An integer specifying the width of the widget in columns, where each column is the width of the character 'M'. This variable will take effect only if the `dim` variable for the widget is set to `nil`.

`cursor_position`

An integer specifying the number of characters to the left of the cursor. Default is `-1`.

`edit_mask`

This variable is not currently implemented.

`max_length`

An integer specifying the maximum number of characters the widget will accept.

`text_flags`

This instance variable controls certain characteristics of the widget, and may be a combination of zero or more of the following flags:

Constant	Description
Pt_CURSOR_VISIBLE	Makes the cursor as visible. Default is ON.
Pt_EDITABLE	Makes the text editable. Default is ON.
Pt_INSERT_MODE	Toggles between insert and replace modes. Default is insert.
Pt_TEXT_AUTO_HIGHLIGHT	Highlights text when widget has focus. Default is OFF.
Pt_CHANGE_ACTIVATE	Internal informational bit.
Pt_EDIT_ACTIVATE	Internal informational bit.

Constant	Description
Pt_TEXT_FULL	Internal informational bit.
Pt_NO_RANGE_SELECTION	Internal informational bit.
Pt_RESIZE_WIDTH	Internal informational bit.
Pt_TEXT_RANGE_ACTIVE	Internal informational bit.
Pt_TEXT_CHANGED	Internal informational bit.

Callbacks

The following callbacks are associated with this widget:

Callback	Description
Pt_CB_MODIFY_VERIFY	This callback is generated when a text string is going to be changed.
Pt_CB_TEXT_CHANGED	This callback is generated when the text string value changes (same as modify_notify.
Pt_CB_MODIFY_NOTIFY	This callback is generated when the text string value changes (same as text_changed.
Pt_CB_MOTION_VERIFY	This callback is generated when the cursor position is going to be changed.
Pt_CB_MOTION_NOTIFY	This callback is generated when the cursor position has changed.

Associated Classes

PtComboBox, PtCallbackInfo, PtTextCallback

Convenience Functions

Arguments

widget	A PtText widget.
start	The start of the selected text.
end	The end of the selected text.

Functions

These functions are extensions of QNX Photon functions. You can refer to the PtText function documentation in QNX Helpviewer for more information about them.

numchars = PtTextGetSelection (widget) -- gets a range of text previously selected with PtTextSetSelection.

Returns the number of characters selected, or -1 to indicate that the *widget* is not a PtText widget.

numchars = PtTextSetSelection (widget, start, end) -- Selects text specified by *start* and *end*. If *start* and *end* are equal to each other, or if they are both greater than the number of characters in the *widget*, this function may return 0.

Returns the number of characters selected, or -1 to indicate that the *widget* is not a PtText widget.

Example

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

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

/*
 * This example puts 3 PtText widgets into a window.
 * The first PtText widget allows for text entry.  Its width is
 * specified at 12 columns (M_width), and maximum length is 15
 * characters. The second PtText widget displays the cursor
 * position in the first PtText widget.  The third PtText widget
 * displays the text entered when the Enter key is pressed in the
 * first widget.
 */
require_lisp("PhotonWidgets.lsp");
require_lisp("PhabTemplate.lsp");
PtInit(nil);

wfile = PhabReadWidgetFile (string(_os_, "-WidgetFiles/wgt/text.wgtw"));
window = PhabCreateWidgets(wfile, nil, nil);

win = PhabLookupWidget(window,#text,nil);
tx1 = PhabLookupWidget(window,#PtText1,nil);
tx2 = PhabLookupWidget(window,#PtText2,nil);
tx3 = PhabLookupWidget(window,#PtText3,nil);
ebut = PhabLookupWidget(window,#TextButtonExit,nil);

/*
 * Specify the characteristics of the PtText widgets.
 */
tx1.dim = nil;
tx1.columns = 12;
tx1.max_length = 15;
tx2.text_flags = cons(Pt_EDITABLE, nil);
tx3.text_flags = cons(Pt_EDITABLE, nil);

/*
 * Attach callbacks for text changes, cursor motion, and pressing
 * the Enter key.
 */
PtAttachCallback(tx1, Pt_CB_TEXT_CHANGED,
                 #entry = tx1.text_string);

PtAttachCallback(tx1, Pt_CB_MOTION_NOTIFY,
                 #tx2.text_string = string(tx1.cursor_position));

PtAttachCallback(tx1, Pt_CB_ACTIVATE, #tx1.text_string = "");
PtAttachCallback(tx1, Pt_CB_ACTIVATE, #tx2.text_string = "");
PtAttachCallback(tx1, Pt_CB_ACTIVATE, #tx3.text_string = entry);
PtAttachCallback(ebut, Pt_CB_ACTIVATE, #exit_program(1));

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

PtToggleButton

`PtToggleButton` — A button that toggles ON or OFF.

Synopsis

```
class PtToggleButton PtLabel
{
    indicator_color;     // color  (Pt_ARG_INDICATOR_COLOR)
    indicator_depth;     // unsigned short  (Pt_ARG_INDICATOR_DEPTH)
    indicator_height;    // unsigned short  (Pt_ARG_INDICATOR_HEIGHT)
    indicator_type;      // unsigned char  (Pt_ARG_INDICATOR_TYPE)
    indicator_width;     // unsigned short  (Pt_ARG_INDICATOR_WIDTH)
    set_color;           // color  (Pt_ARG_SET_COLOR)
    set_fill;            // unsigned char  (Pt_ARG_SET_FILL)
    spacing;             // unsigned short  (Pt_ARG_SPACING)
}
```

Base Classes

`PtWidget` <-- `PtBasic` <-- `PtLabel` <-- `PtToggleButton`

Derived Classes

`PtOnOffButton`

Description

This widget is a button that toggles ON and OFF, made up of an on/off indicator and a text label. Its instance variables control the size, shape, and color of the on/off indicator, as well as the font size of the corresponding label text.

For detailed information, please refer to PtToggleButton in the Photon documentation.

Instance Variables

`indicator_color`

A number specifying the color of the on/off indicator when the button is OFF. Default is `0xc0c0c0` (grey).

`indicator_depth`

A number of pixels specifying the border width of the on/off indicator (giving the appearance of depth when the button is ON). Default is `2`.

`indicator_height`

A number of pixels specifying the height of the on/off indicator. Default is adjusted according to the font size of the label text.

`indicator_type`

This instance variable controls the shape of the indicator, and may have one of the following values:

Constant	Description
`Pt_N_OF_MANY`	Gives the on/off indicator a diamond shape.
`Pt_ONE_OF_MANY`	Gives the on/off indicator a square shape.
`Pt_RADIO`	Makes the on/off indicator a round, radio-button style.

Constant	Description
`Pt_ROUND`	Gives the on/off indicator a round shape.

`indicator_width`

A number of pixels specifying the width of the on/off indicator. Default is adjusted according to the font size of the label text.

`set_color`

A number indicating the color of the indicator when the button is ON (pressed in). Default is `0xaaaaaa` (dark grey).

`set_fill`

A control bit for `set_color`. The default, 1, means that the color specified in `set_color` will be used for the on/off indicator when the button is ON. If 0 is specified here, the on/off indicator will not change color when the button is ON.

`spacing`

A number of pixels specifying the space between the on/off indicator and the label text. Default is `4`.

PtTree

PtTree — An expanding/collapsing tree-style list of items.

Synopsis

```
class PtTree PtGenTree
{
    flags;          // flag
    tree_images;    // PhImage array  (Pt_ARG_TREE_IMAGES)
    tree_imgmask;   // unsigned long  (Pt_ARG_TREE_IMGMASK)
}
```

Base Classes

PtWidget <-- PtBasic <-- PtContainer <-- PtGenList <-- PtGenTree <-- PtTree

Description

This widget displays items in a tree-style list, such as in a file listing or table of contents. Each item has a string for displaying its name, and two images to indicate whether it is currently expanded or collapsed.

For detailed information, please refer to PtTree in the Photon documentation.

Instance Variables

flags

This is a Gamma extension which has no corresponding Photon resource. It is not yet documented.

tree_images

An array of two PhImages used to show the expanded/collapsed state of tree items.

tree_imgmask

This instance variable defines the image selection mask, and may have one of the following values:

Constant	Description
Pt_LIST_ITEM_SELECTED	The item is selected.
Pt_LIST_ITEM_CURRENT	The item is the current item.
Pt_TREE_ITEM_EXPANDABLE	The item can be expanded to show its branches.
Pt_TREE_ITEM_EXPANDED	The item is expanded.

Callbacks

The following callbacks are associated with this widget:

Callback	Description
Pt_CB_TREE_STATE	This callback is generated when the state of the tree changes (expands/collapses).
Pt_CB_TREE_SELECTION	This callback is generated when a tree item is selected.

Associated Classes

`PtTreeItem`, `PtTreeCallback`, `PtCallbackInfo`

Convenience Functions

Arguments

brother	A `PtTreeItem` at the same level as another `PtTreeItem`.
child	A `PtTreeItem` one level below another `PtTreeItem`.
event	An event to be passed to the tree_state callback.
images	A list of PhImage structures.
indexes	A sorted list or array of indexes.
item	A `PtTreeItem`.
parent	A `PtTreeItem` one level above another `PtTreeItem`.
sel_image	The index of the image to be displayed when the `PtTreeItem` is set.
string	The string resource of a `PtTreeItem`.
unsel_image	The index of the image to be displayed when the `PtTreeItem` is not set.
widget	A `PtTree` widget.

Functions

Most of these functions are extensions of QNX Photon functions. You can refer to the PtTree function documentation in QNX Helpviewer for more information about them.

`PtTreeAddAfter` `(widget, item, brother)` -- inserts a tree item below the *`brother`* item in the *`widget`*.

Returns `t`.

`PtTreeAddFirst` `(widget, item, brother)` -- inserts an item as a child item, directly below the *`parent`* item in the *`tree`*, in front of any existing children items. If *`parent`* is `nil`, the *`item`* will be placed at the root level of the tree, before all other items. The *`tree`* argument can be `nil`, as long as *`parent`* is not attached to any tree widget.

Returns `t`.

`index = PtTreeAddImages` `(widget, images)` -- adds the images in the *`images`* list to the *`widget`*'s image list. The added images are indexed starting with the last of the *`widget`*'s existing images. The *`widget`* makes a copy of the PhImage structures of *`images`*, but it doesn't copy the palettes or image data of the structure members.

Returns the index of the first of the added images on success, or `nil` on failure.

`array = PtTreeAllItems` `(widget)` -- fills an array with all items in the *`widget`*. Items that belong to collapsed subtrees aren't included in the array.

Returns an array.

`item = PtTreeAllocItem` `(widget, string, sel_image, unsel_image)` -- allocates a new item added by `PtTreeAddFirst` or `PtTreeAddAfter`. Images in *`sel_img`* and *`unsel_img`* are displayed as the item is set or not. (An item is considered set when its flags masked

with tree_imgmask are not nil.) These images must already be present in the widget. If an index passed to PtTreeAllocItem is higher than the current image count, it will be changed to -1, meaning "no image."

Returns a PtTreeItem.

PtTreeClearSelection (widget) -- clears a selection.

Returns t.

PtTreeCollapse (widget, item, event) -- collapses the *item* subtree. The *widget* tree must contain the *item*, or be nil if the *item* doesn't belong to any tree. If there is a tree, the tree_state callback is invoked, and the *event* is passed to it.

Returns t.

PtTreeExpand (widget, item, event) -- expands the *item* subtree. The *widget* tree must contain the *item*, or be nil if the *item* doesn't belong to any tree. If there is a tree, the tree_state callback is invoked, and the *event* is passed to it. If the callback function prevents expansion by setting the callback expand field to a non-zero value, that value will be returned.

Returns 0 on success, else a value that prevented the expansion.

PtTreeFreeAllItems (widget) -- unlinks and frees all items in the *widget* tree. It is automatically called whenever a PtTree widget is deleted.

Returns t.

PtTreeFreeItems (item) -- frees the *item*, its brothers, and its subtrees. The *item* must not belong to any tree, or else it must be removed first with a call to PtTreeRemoveChildren, PtTreeRemoveItem, or PtTreeRemoveList.

Returns t.

item = PtTreeGetCurrent (widget) -- gets the current item from the *widget* tree.

Returns the current item.

array = PtTreeGetSelIndexes (widget) -- makes an index array of all the selected items in the *widget*. Items belonging to collapsed subtrees are not indexed in the array.

Returns an array of indexes.

integer = PtTreeGoto (widget, item) -- sets the current item and (if necessary) the current position so that the new current item is visible. If *item* is nil, there will be no current item. A non-zero return value means the *item* belongs to a collapsed subtree, and a callback function prevented the subtree from being expanded.

Returns 0 on success, else a non-zero integer.

item = PtTreeItemBrother (item) -- gets the first brother of an item.

Returns the first brother of the *item*, or nil.

item = PtTreeItemFather (item) -- gets the father of an item.

Returns the father of the *item*, or nil.

index = PtTreeItemIndex (item) -- calculates the index of the *item* in the tree. The index of the first item is 1.

Returns the index of the *item*, or 0 if it belongs to a collapsed subtree.

item = PtTreeItemSon (item) -- gets the first son of an item.

Returns the first son of the *item*, or nil.

item = PtTreeModifyItem (widget, item, string, sel_image, unsel_image) -- modifies the *item* according to the *string*, *sel_img*, and *usel_img* passed to it. If *string* is nil, the *item*'s string resource won't be changed.

Returns the *item*, modified.

Don't use this function to modify an item that hasn't yet been added to the tree.

item = PtTreeRemoveChildren (item) -- unlinks all the children of the *item* and returns the first child. They can then all be freed by a call to PtTreeFreeItems on this child.

Returns the first child, a PtTreeItem.

This function will return nil if any children are expanded. All children must be collapsed by a call to PtTreeCollapse before this function will work.

PtTreeRemoveItem (widget, item) -- unlinks the *item* and all its children from its parent and brothers. If the *item* doesn't belong to a tree, *widget* must be set to nil. If *widget* is nil, and the *item* has no parent, but has a previous brother, the function will unlink the *item* from its brother. This function never clears the tree_item_expandable flag in the *item*'s parent.

Returns t.

PtTreeRemoveList (widget, item) -- strips off the entire bottom of the tree, starting at the *item*, taking all subsequent brothers and children, setting their parent resources to nil. If the *item* has no parent, nil can be passed for *widget*, but if a parentless *item* does have a previous brother, the function won't be able to find it or unlink the *item* from it.

Returns t.

item = PtTreeRootItem (widget) -- gets the first root item of a the *widget* tree.

Returns the first root item, or nil.

PtTreeSelect (widget, item) -- selects the *item*.

Returns t.

array = PtTreeSelectedItems (widget) -- makes an array and fills it with the items of the *widget*. Those items that belong to collapsed subtrees aren't included in the array.

Returns an array of items.

integer = PtTreeSetSelIndexes (widget, indexes) -- sets the selection indexes. The array or list of *indexes* must be sorted. Otherwise the number of items returned may be smaller than the actual size of the list or array.

Returns the number of indexes.

PtTreeShow (widget, item) -- sets the current position so the *item* is visible. If *item* is nil, the function does nothing.

Returns 0 on success, else a non-zero value, indicating that the *item* belongs to a collapsed subtree which could not be expanded.

PtTreeUnselect (widget, item) -- unselects the *item*.

Returns t.

PtTreeUnselectNonBrothers (widget, item) -- unselects all items that are not brothers of the *item*. If you pass nil for *item*, the current item is used.

Returns t.

PtTrend

PtTrend — A graph for displaying real-time trends (in QNX 6).

Synopsis

```
class PtTrend PtBasic
{
    trend_data;        // array of arrays of integer  (Pt_ARG_TREND_DATA)
    trend_flags;       // flag  (Pt_ARG_TREND_FLAGS)
    trend_attributes;    // integer array  (Pt_ARG_TREND_ATTRIBUTES)
    trend_color_list;    // color array  (Pt_ARG_TREND_COLOR_LIST)
    trend_count;         // integer  (Pt_ARG_TREND_COUNT)
    trend_grid_color;    // color  (Pt_ARG_TREND_GRID_COLOR)
    trend_grid_x;        // short  (Pt_ARG_TREND_GRID_X)
    trend_grid_y;        // short  (Pt_ARG_TREND_GRID_Y)
    trend_inc;           // short  (Pt_ARG_TREND_INC)
    trend_max;           // short  (Pt_ARG_TREND_MAX)
    trend_min;           // short  (Pt_ARG_TREND_MIN)
}
```

Base Classes

PtWidget <-- PtBasic <-- PtTrend

Description

This widget is a real-time trend display that can plot multiple trends and display them in a moving format, to show changes over time. The QNX 4 version of this widget is RtTrend.

For detailed information, please refer to PtTrend in the Photon documentation.

Instance Variables

pttrend_data

An array of data arrays. Each data array corresponds to one trend line on the display.

pttrend_flags

Flags that specify grid characteristics and direction. If any of these flags is set incorrectly, the widget will not display any trends.

The grid options are only supported by 8-bit graphics environments, and only on some graphics cards.

This instance variable may be a combination of zero or more of the following flags:

Constant	Description
Pt_GRID	Draws a grid. This flag requires exactly one of Pt_GRID_IS_TRANSLUCENT, Pt_GRID_ABOVE_TRENDS, or Pt_TRENDS_ABOVE_GRID to be set as well.
Pt_GRID_IS_TRANSLUCENT	Makes the grid that was set using Pt_GRID appear to be translucent.
Pt_GRID_ABOVE_TRENDS	Makes the grid that was set using Pt_GRID appear to be superimposed over the trends.
Pt_GRID_FORCE	Forces drawing the grid, despite possible flickering effect due to poor hardware support.

Constant	Description
Pt_PIXEL	The grid is drawn using a pixel array instead of vector graphics.
Pt_TRENDS_ABOVE_GRID	Makes the grid that was set using Pt_GRID appear to be underneath the trends.
Pt_TREND_HORIZONTAL	Causes the trend to move horizontally. This flag requires exactly one of Pt_TREND_LEFT_TO_RIGHT or Pt_TREND_RIGHT_TO_LEFT.
Pt_TREND_VERTICAL	Causes the trend to move vertically. This flag requires exactly one of Pt_TREND_TOP_TO_BOTTOM or Pt_TREND_BOTTOM_TO_TOP.
Pt_LEFT_TO_RIGHT	Sets the direction for Pt_TREND_HORIZONTAL.
Pt_RIGHT_TO_LEFT	Sets the direction for Pt_TREND_HORIZONTAL.
Pt_TOP_TO_BOTTOM	Sets the direction for Pt_TREND_VERTICAL.
Pt_BOTTOM_TO_TOP	Sets the direction for Pt_TREND_VERTICAL.

`trend_attributes`

An array that indexes the `trend_color_list`. Index numbers are consecutive integers starting with 0, and each number points to an element of the `trend_color_list`. Default is `nil`.

`trend_color_list`

An array of colors to be used for plotting trend lines. Default array has a single number: [16711680], which equals 0xff0000 (red).

`trend_count`

An integer specifying the number of trends.

`trend_grid_color`

A number specifying the color of the grid to be used if the `Pt_GRID` flag is set in the `pttrend_flags` variable. Default is `0xc0c0c0` (grey).

`trend_grid_x`

A number specifying the number of vertical grid lines to be used if the `Pt_GRID` flag is set in the `pttrend_flags` variable. Default is `5`.

`trend_grid_y`

A number specifying the number of horizontal grid lines to be used if the `Pt_GRID` flag is set in the `rttrend_flags` variable. Default is `5`.

`trend_inc`

A number specifying the spacing of the plotted points along the moving axis. Default is `1`.

`trend_max`

A number specifying the maximum value for the display. Default is `32,767`.

`trend_min`

A number specifying the maximum value for the display. Default is `-32,768`.

Example

Here is a very simple example of an PtTrend widget. For more complex examples, see the Gamma demos **cpumem** and **trend**. This example, ex_Trend.g, is included in the product distribution.

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

/*
 * This example illustrates an RtTrend in QNX 4, or a PtTrend in QNX 6,
 * by creating and displaying a sine function.
 */
require_lisp("PhotonWidgets.lsp");
require_lisp("PhabTemplate.lsp");
PtInit(nil);

wfile = PhabReadWidgetFile (string(_os_, "-WidgetFiles/wgt/trend.wgtw"));
window = PhabCreateWidgets(wfile, nil, nil);
win = PhabLookupWidget(window,#trend,nil);
tr = PhabLookupWidget(window,#Trend,nil);
ebut = PhabLookupWidget(window,#TrendExitButton,nil);
PtAttachCallback(ebut, Pt_CB_ACTIVATE, #exit_program(1));

Values := make_array (1);
valarray = make_array (1);

/*
 * Make a function to generate a trend.
 */

Sinex = 0;

function Sine (inc, max, min)
{
 local  temp = sin (Sinex), diff = (max - min) / 2;
 Sinex = Sinex + inc;
 temp * diff + min + diff;
}

Trend = list (Sine, 0.07, 550, 250);

/*
 * Make a function to evaluate the trend.
 */

function FillTrend ()
{
  valarray[0] = eval(Trend);
  Values[0] = valarray;
  if (_os_ == "QNX4")
    tr.rttrend_data = Values;
  else if (_os_ == "QNX6")
    tr.trend_data = Values;
}

/*
 * Set up the trend.
 */

tr.trend_max = 800;
tr.trend_min = 0;

/*
 * Start an update timer, and loop forever.
 */

every (.03, #FillTrend());

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

PtTty

PtTty — A terminal that is attached to a device.

Synopsis

```
class PtTty PtTerminal
{
    tty_buflen;         // unsigned short  (Pt_ARG_TTY_BUFLEN)
    tty_cmd;            // string  (Pt_ARG_TTY_CMD)
    tty_devsize;        // PhPoint  (Pt_ARG_TTY_DEVSIZE)
    tty_exit_status;    // integer  (Pt_ARG_TTY_EXIT_STATUS)
    tty_fd;             // integer  (Pt_ARG_TTY_FD)
    tty_fdset;          // unsigned short  (Pt_ARG_TTY_FDSET)
    tty_flags;          // flag  (Pt_ARG_TTY_FLAGS)
    tty_input;          // string  (Pt_ARG_TTY_INPUT)
    tty_mfd;            // integer  (Pt_ARG_TTY_MFD)
    tty_path;           // string  (Pt_ARG_TTY_PATH)
    tty_pid;            // long  (Pt_ARG_TTY_PID)
    tty_pri;            // integer  (Pt_ARG_TTY_PRI)
    tty_pseudo;         // string  (Pt_ARG_TTY_PSEUDO)
}
```

Base Classes

PtWidget <-- PtBasic <-- PtContainer <-- PtTerminal <-- PtTty

Description

This widget is a child of PtTerminal, but can be attached to a device, and has the capability of device I/O.

For detailed information, please refer to PtTty in the Photon documentation.

Instance Variables

tty_buflen

A number specifying the length of a buffer for reading data from the device. Default is 1024.

tty_cmd

A write-only string used to spawn a user's log-in shell on the device. Default is nil.

tty_devsize

A PhPoint specifying the size (in rows and columns) of the current device, which may be different from the size of the widget. Default is:

```
{PhPoint (x . 0) (y . 0)}
```

tty_exit_status

A number specifying the exit status of a spawned process.

tty_fd

A number that specifies a file descriptor to which the widget gets attached. The default, -1, breaks the attachment, or indicates no attachment.

tty_fdset

A number that acts as a bitmask to define file descriptors for new processes. The default, 7, makes available descriptors 0, 1, and 2.

`tty_flags`

This instance variable controls sizing and other characteristics, and may be a combination of zero or more of the following flags:

Constant	Description
`Pt_TTY_TERMRESIZE`	Sets the terminal widget size when the device size changes.
`Pt_TTY_DEVRESIZE`	Resizes the device when the terminal size changes.
`Pt_TTY_DEVLIMIT`	Resizes the device when its size changes beyond the size limits of the terminal.
`Pt_TTY_DEVFORCE`	Resizes the device size to the terminal's size.
`Pt_TTY_SETENV`	Set or modify environment variables.
`Pt_TTY_ARGV0`	Use the program name as argv[0].
`Pt_TTY_BUF_PRIVATE`	The widget is using an allocated buffer.

`tty_input`

A string comprising the input to be written.

`tty_mfd`

The "master" file descriptor, which is a number equal to the `tty_fd`, or else to the file descriptor of the pty if `tty_pseudo` was used to open the device.

`tty_path`

A string specifying a pathname for a reading and writing to a device. Default is `nil`.

`tty_pid`

A number specifying the process ID of the process spawned on the device. Default is `0`.

`tty_pri`

A number specifying Photon pulse priority. Default is `-1`.

`tty_pseudo`

A string specifying the name of an unused pseudo-tty device to attach to. The default, `nil`, specifies that //0/dev be used. An empty string "" specifies that /dev be used.

Callbacks

The following callbacks are associated with this widget:

Callback	Description
`Pt_CB_TTY_DEVSIZE`	This callback is generated when a resize event is received, but before the terminal widget is resized.
`Pt_CB_TTY_OUTPUT`	This callback is generated when output is received, before passing it on to the terminal widget.
`Pt_CB_TTY_TERMINATED`	This callback is generated when a child process (terminal) has been terminated.

Associated Classes

PtTtyOutput, PtTerminalSizeChange, PtCallbackInfo

PtUpDown

PtUpDown — A pair of buttons that increment and decrement a value.

Synopsis

```
class PtUpDown PtContainer
{
    updown_arm_data_bottom;        // PhImage  (Pt_ARG_UPDOWN_ARM_DATA_BOTTOM)
    updown_arm_data_left;          // PhImage  (Pt_ARG_UPDOWN_ARM_DATA_LEFT)
    updown_arm_data_right;         // PhImage  (Pt_ARG_UPDOWN_ARM_DATA_RIGHT)
    updown_arm_data_top;           // PhImage  (Pt_ARG_UPDOWN_ARM_DATA_TOP)
    updown_bottom_border_color;    // color  (Pt_ARG_UPDOWN_BOTTOM_BORDER_COLOR)
    updown_data_bottom;            // PhImage  (Pt_ARG_UPDOWN_DATA_BOTTOM)
    updown_data_left;              // PhImage  (Pt_ARG_UPDOWN_DATA_LEFT)
    updown_data_right;             // PhImage  (Pt_ARG_UPDOWN_DATA_RIGHT)
    updown_data_top;               // PhImage  (Pt_ARG_UPDOWN_DATA_TOP)
    updown_fill_color;             // color  (Pt_ARG_UPDOWN_FILL_COLOR)
    updown_flags;                  // flag  (Pt_ARG_UPDOWN_FLAGS)
    updown_highlight_round;        // unsigned short  (Pt_ARG_UPDOWN_HIGHLIGHT_ROUND)
    updown_margin_height;          // unsigned short  (Pt_ARG_UPDOWN_MARGIN_HEIGHT)
    updown_margin_width;           // unsigned short  (Pt_ARG_UPDOWN_MARGIN_WIDTH)
    updown_orientation;            // integer  (Pt_ARG_UPDOWN_ORIENTATION)
    updown_spacing;                // unsigned integer  (Pt_ARG_UPDOWN_SPACING)
    updown_top_border_color;       // color  (Pt_ARG_UPDOWN_TOP_BORDER_COLOR)
}
```

Base Classes

PtWidget <-- PtBasic <-- PtContainer <-- PtUpDown

Description

This widget is a pair of horizontal or vertical buttons that are used to increment and decrement values. The button appearance and images can be modified in a variety of ways.

For detailed information, please refer to PtUpDown in the Photon documentation.

Instance Variables

`updown_arm_data_bottom`
`updown_arm_data_left`
`updown_arm_data_right`
`updown_arm_data_top`

A `PhImage` to be displayed on the bottom, left, right, or top button when it is pressed.

`updown_bottom_border_color`

A number specifying the color of the bottom and right borders. Default is `0x0` (black).

`updown_data_bottom`
`updown_data_left`
`updown_data_right`
`updown_data_top`

A `PhImage` to be displayed on the bottom, left, right, or top button.

`updown_fill_color`

A number specifying the color of the space between the buttons (if any) and/or of the margins (if any). Default is `0xffffffff` (transparent).

`updown_flags`

This instance variable controls various characteristics of the widget, and may be a combination of zero or more of the following flags:

Constant	Description
`Pt_HIGHLIGHTED`	Gives the widget a beveled or etched border highlight (usually beveled).
`Pt_ETCH_HIGHLIGHT`	Gives the widget an etched border highlight.
`Pt_CLIP_HIGHLIGHT`	Cuts off the corners of the highlight rectangle.
`Pt_BLOCKED`	Blocks interaction with Photon events for widget and non-window children.
`Pt_MENUABLE`	Enables menu callbacks for right-button mouse clicks.
`Pt_GETS_FOCUS`	Permits focusing on widget, as appropriate.

`updown_highlight_round`

A number of pixels specifying the radius for rounding the corners of the margins. Default is `0`. The corners of the buttons themselves do not get rounded using this variable.

`updown_margin_height`
`updown_margin_width`

A number of pixels specifying a margin (height or width) around the widget. Default is `0`.

`updown_orientation`

This instance variable may have one of the following values:

Constant	Description
`Pt_VERTICAL`	The arrows point vertically (the default).
`Pt_HORIZONTAL`	The arrows point horizontally.

`updown_spacing`

A number of pixels specifying the distance to separate the buttons. Default is `0`.

`updown_top_border_color`

A number specifying the color of the top and left borders.

PtWidget

PtWidget — The parent class of all widgets.

Synopsis

```
class PtWidget
{
    area;            // PhArea  (Pt_ARG_AREA)
    border_width;    // unsigned short  (Pt_ARG_BORDER_WIDTH)
    cursor_color;    // color  (Pt_ARG_CURSOR_COLOR)
    cursor_type;     // integer  (Pt_ARG_CURSOR_TYPE)
    dim;             // PhDim  (Pt_ARG_DIM)
    eflags;          // flag  (Pt_ARG_EFLAGS)
    flags;           // flag  (Pt_ARG_FLAGS)
    help_topic;      // string  (Pt_ARG_HELP_TOPIC)
    pos;             // PhPoint  (Pt_ARG_POS)
    resize_flags;    // flag  (Pt_ARG_RESIZE_FLAGS)
    rid;             // region ID  (RID)
}
```

Derived Classes

PtBasic

Description

This widget is the parent class of all widgets, and contains many commonly-used instance variables. The PtWidget itself is rarely instantiated.

For detailed information, please refer to PtWidget in the Photon documentation.

Instance Variables

area

A PhArea instance, specifying the point of origin and dimensions of the widget. The value of this variable should only be changed with the SetArea method (see below).

border_width

A number of pixels specifying the border width. Default is 2, maximum is 15.

cursor_color

A number specifying the color of the pointer when it is inside the widget. Default is 0xffffe0 (cream).

cursor_type

The type of cursor.

This instance variable may have one of the following values:

Constant	Description
Ph_CURSOR_NO_INHERIT	Do not inherit the parent's cursor.
Ph_CURSOR_MANUAL_CONTROL	Not implemented.
Ph_CURSOR_NONE	Do not show a cursor.
Ph_CURSOR_BITMAP	Not implemented.
Ph_CURSOR_INHERIT	Inherit the parent's cursor.

Constant	Description
Ph_CURSOR_POINTER	Pointer
Ph_CURSOR_BIG_POINTER	Big Pointer
Ph_CURSOR_MOVE	Move
Ph_CURSOR_CROSSHAIR	Crosshair
Ph_CURSOR_CLOCK	Clock
Ph_CURSOR_WAIT	Wait
Ph_CURSOR_NOINPUT	No Input
Ph_CURSOR_DONT	Don't
Ph_CURSOR_FINGER	Finger
Ph_CURSOR_INSERT	Insert
Ph_CURSOR_DRAG_VERTICAL	Drag Vertical
Ph_CURSOR_DRAG_TOP	Drag Top
Ph_CURSOR_DRAG_BOTTOM	Drag Bottom
Ph_CURSOR_DRAG_HORIZONTAL	Drag Horizontal
Ph_CURSOR_DRAG_LEFT	Drag Left
Ph_CURSOR_DRAG_RIGHT	Drag Right
Ph_CURSOR_DRAG_BACKDIAG	Drag Backdiag
Ph_CURSOR_DRAG_TL	Drag Top Left
Ph_CURSOR_DRAG_BR	Drag Bottom Right
Ph_CURSOR_DRAG_FOREDIAG	Drag Foreward Diagonal
Ph_CURSOR_DRAG_TR	Drag Top Right
Ph_CURSOR_DRAG_BL	Drag Bottom Left
Ph_CURSOR_POINT_WAIT	Point Wait
Ph_CURSOR_LONG_WAIT	Long Wait
Ph_CURSOR_QUESTION_POINT	Question Point
Ph_CURSOR_PASTE	Paste

`dim`

A PhDim instance specifying the widget dimensions. The value of this variable should only be changed with the `SetDim` method (see below).

`eflags`

Extended flags common to all widgets.

This instance variable may be a combination of zero or more of the following flags:

Constant	Description
Pt_CONSUME_EVENTS	Consume all events, even if no event handler is defined
Pt_INTERNAL_HELP	Display the widget's help in a balloon instead of in the Helpviewer
Pt_WIN_REQUEST	Internal informational bit
Pt_SKIP_CHILDREN	Internal informational bit

Constant	Description
Pt_DAMAGE_PARENT	Propagate any damage on this widget to its parent
Pt_DAMAGE_ON_FOCUS	Internal informational bit

flags

Flags common to all widgets.

This instance variable may be a combination of zero or more of the following flags:

Constant	Description
Pt_HIGHLIGHTED	Draw a beveled border around the widget.
Pt_AUTOHIGHLIGHT	Automatically display/remove the highlight border as the cursor passes over the widget.
Pt_ETCH_HIGHLIGHT	Draw a double bevel if Pt_HIGHLIGHTED is on.
Pt_SET	Make the widget 'set'. This will invert the coloring on the bevel as well.
Pt_TOGGLE	The widget's SET flag will toggle on each mouse click instead of changing with mouse down/mouse up events.
Pt_SELECTABLE	The widget may be selected, causing Pt_CB_ARM/ACTIVATE/DISARM events.
Pt_GHOST	The widget is displayed 'ghosted'. This does not affect its response to events.
Pt_BLOCKED	The widget will be unresponsive to events.
Pt_REALIZED	The widget is visible on the display.
Pt_CLIP_HIGHLIGHT	The corners of the highlight rectangle are clipped off.
Pt_OPAQUE	Makes the widget opaque within its own extent and for everything behind it.
Pt_DELAY_REALIZE	The widget won't be realized except by a call to PtRealizeWidget().
Pt_GETS_FOCUS	The widget may get the keyboard focus.
Pt_MENU_BUTTON	Makes the widget a menu item.
Pt_DESTROYED	Marks the widget for destruction.
Pt_DAMAGED	Marks the widget for repairs.
Pt_OBSCURED	The widget is covered by another widget, or is outside its parent's canvas.
Pt_IN_FLUX	A call to PtContainerHold() has been made on the widget.
Pt_CLEAR	Keeps the widget's extent clear of any brothers in front of it.
Pt_DAMAGE_FAMILY	The widget and its children will be repaired.
Pt_SELECT_NOREDRAW	The widget will not redraw when it is selected.

Constant	Description
Pt_WIDGET_REBUILD	The widget will be rebuilt once all resources have been changed.
Pt_WIDGET_RESIZE	The widget will be resized once all resources have been changed.
Pt_PROCREATED	The widget is a procreated child of a compound widget.
Pt_ALL_BUTTONS	The widget treats events on any mouse button as a selection.
Pt_FOCUS_RENDER	The widget will attempt to indicate that it has the keyboard focus through some means.
Pt_CALLBACKS_ACTIVE	Callbacks for this widget will be called due to changes through code, not just due to user interactions.
Pt_MENUABLE	This widget will respond to the menu button with a Pt_CB_MENU event.
Pt_NOREDRAW_SET	Noredraw Set
Pt_FREE_MEMORY	Frees memory associated with widget pointers.
Pt_REGION	Force the widget to have a region ID.
Pt_REALIZING	The widget is being realized.

help_topic

A string that contains help information, if the eflags Pt_INTERNAL_HELP flag is set. Otherwise the variable sets the topic position in the HTML help file.

pos

A PhPoint that specifies the position of the upper-left corner of the widget. The value of this variable should only be changed with the SetPos method (see below).

resize_flags

Flags common to all widgets that control their resizability on the x and/or y axis.

This instance variable may be a combination of zero or more of the following flags:

Constant	Description
Pt_RESIZE_X_INITIAL	Only resize the widget in X to contain its children when first created.
Pt_RESIZE_X_AS_REQUIRED	Resize the widget in X only if its children extend beyond the current X extent
Pt_RESIZE_X_ALWAYS	Always resize the widget to exactly contain its children.
Pt_RESIZE_Y_INITIAL	Only resize the widget in X to contain its children when first created.
Pt_RESIZE_Y_AS_REQUIRED	Resize the widget in Y only if its children extend beyond the current Y extent
Pt_RESIZE_Y_ALWAYS	Always resize the widget to exactly contain its children.

`rid`

The region ID of the widget, or 0 if the widget has no explicit region. The region can be made explicit using the `Pt_REGION` flag.

The region ID is read-only. Do not attempt to set this variable to any other value.

Callbacks

The following callbacks are associated with this widget:

Callback	Description
`Pt_CB_BLOCKED`	This callback is generated when a blocked event must be ignored.
`Pt_CB_DESTROYED`	This callback is generated when a widget is destroyed.
`Pt_CB_HOTKEY`	This callback is generated when a key event matches key cap and key modifiers of a common class.
`Pt_CB_RAW`	This callback is generated when a raw event matches the event mask of a raw callback.
`Pt_CB_REALIZED`	This callback is generated when a widget is realized.
`Pt_CB_UNREALIZED`	This callback is generated when a widget is unrealized.

Associated Classes

`PtHotkeyCallback`, `PtCallbackInfo`

Methods

To use any of these methods, you must load the PhotonWidgets.lsp library with a call to `require_lisp`("PhotonWidgets.lsp").

Setting Position, Dimension, and Area

The three methods here are the only reliable way to set or change the `dim`, `pos`, and `area` variables of any widget. Attempting to change these variables or their sub-components without using these methods will lead to unpredictable results in your program.

Syntax

```
widget.SetPos(x,y)
widget.SetDim(w,h)
widget.SetArea(x,y,w,h)
```

Arguments

x	The integer number of pixels on the x-axis (to the right) to locate the widget, measuring from the upper-left corner of the containing widget.
y	The integer number of pixels on the y-axis (down) to locate the widget, measuring from the upper-left corner of the containing widget.
w	The width of the widget in pixels, expressed as an integer.
h	The height of the widget in pixels, expressed as an integer.

Return

`t` on success, or `nil` on failure.

Description

All widget measurements are made in pixels, with respect to the upper-left corner of the widget. The position of a widget is the distance of its upper-left corner from the upper-left corner of its immediate container, on the x and y axes. The dimensions of a widget are always relative to its own upper-left corner. Remember to load the PhotonWidgets.lsp library with a call to **`require_lisp("PhotonWidgets.lsp")`** to use these methods.

Example

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

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

/*
This example puts up a window with a pane and a button, with the
relevant positions and dimensions specified.
*/

function print_area (widget)
{
  princ("\nThe area of the ", class_name(widget),
  " widget is: \n", widget.area, "\n");
}

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

win = new(PtWindow);
win.SetDim(300,200);
win.title = "PtWidget.SetArea Example";

pane = new(PtPane);
pane.SetArea(25,25,250,150);
pane.fill_color = PgRGB(240,220,100);

but = new(PtButton);
but.SetPos(90,25);
but.text_string = "Press Here";

PtAttachCallback(but, Pt_CB_ACTIVATE, ‘print_area(@pane));
PtAttachCallback(but, Pt_CB_ACTIVATE, ‘print_area(@but));


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

Setting, Testing, and Clearing Bits

Syntax

```
widget.SetBit(ivar,flag)
widget.ClearBit(ivar,flag)
widget.TestBit(ivar,flag)
```

Arguments

ivar	The name of the instance variable whose flag you wish to set, test, or clear.
flag	The flag you wish to set, test, or clear.

Return

`t` on success, or `nil` on failure.

Description

These methods set, clear and test individual flags of instance variables for any PtWidget. Don't forget to load the PhotonWidgets.lsp library with a call to **`require_lisp("PhotonWidgets.lsp")`** in order to use them.

Example

An example of these methods can be found at the end of the tutorial Set, Test, and Clear Widget Flags in the Programmer's Manual.

Copying Widgets

Syntax

```
widget.Copy()
```

Arguments

none

Returns

A copy of the widget, or an error message.

Description

This method copies any PtWidget, giving it the same parentage and variable values as the original, and putting it the same position. The values of the following variables are *not* copied:

```
accel_key
fill_pattern
help_topic
label_balloon
trans_pattern
window_icon
```

Further, `_callbacks` and `handles selected` (neither of which are actually variables) are also not copied.

Remember to load the PhotonWidgets.lsp library with a call to **`require_lisp("PhotonWidgets.lsp")`** before using this method.

Example

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

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

/*
 * This example puts up a window with two buttons: an
 * original and copy, and prints their definitions.
 */

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

win = new(PtWindow);
win.SetDim(200,130);
win.fill_color = PgRGB(200,220,230);

but = new(PtButton);
but.SetArea(50,25,90,30);
but.fill_color = PgRGB(240,220,180);
but.text_string = "Original";

/*
 * Make the copy and print both widget definitions.
 */
but2 = but.Copy();
but2.SetPos(50,70);
but2.text_string = "Copy";

pretty_princ("Original Button: ",but,"\n");
pretty_princ("Copied Button: ",but2,"\n");

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

Convenience Functions

Arguments

widget	A `PtWidget`.

Functions

These functions are extensions of QNX Photon functions. You can refer to the PtWidget function documentation in QNX Helpviewer for more information about them.

`PtWidgetCanvas` (*`widget`*) -- determines the area inside the *`widget`*'s border.

Returns a `PtRect`.

`PtWidgetExtent` (*`widget`*) -- determines the area of the *`widget`*.

Returns a `PtRect`.

`PtWidgetOffset` (*`widget`*) -- determines the offset point (calculated using the upper-left corner) of the parent of the *`widget`* from its parent.

Returns `t`.

PtWindow

PtWindow — A window managed by the Photon Window Manager.

Synopsis

```
class PtWindow PtContainer
{
    cursor_override;          // integer  (Pt_ARG_WINDOW_CURSOR_OVERRIDE)
    default_action;           // PtWidget
    force_front;              // integer
    help_root;                // string  (Pt_ARG_WINDOW_HELP_ROOT)
    icon_window;              // PtWidget  (Pt_ARG_ICON_WINDOW)
    managed_flags;            // flag  (Pt_ARG_WINDOW_MANAGED_FLAGS)
    max_height;               // short  (Pt_ARG_MAX_HEIGHT)
    max_width;                // short  (Pt_ARG_MAX_WIDTH)
    min_height;               // short  (Pt_ARG_MIN_HEIGHT)
    min_width;                // short  (Pt_ARG_MIN_WIDTH)
    notify_flags;             // flag  (Pt_ARG_WINDOW_NOTIFY_FLAGS)
    render_flags;             // flag  (Pt_ARG_WINDOW_RENDER_FLAGS)
    state;                    // flag  (Pt_ARG_WINDOW_STATE)
    title;                    // string  (Pt_ARG_WINDOW_TITLE)
    title_color;              // color  (Pt_ARG_WINDOW_TITLE_COLOR)
    window_active_color;      // color  (Pt_ARG_WINDOW_ACTIVE_COLOR)
    window_inactive_color;    // color  (Pt_ARG_WINDOW_INACTIVE_COLOR)
}
```

Base Classes

PtWidget <-- PtBasic <-- PtContainer <-- PtWindow

Derived Classes

PtIcon

Description

This widget is a Photon window, with a title bar, resizing buttons, and basic control menu.

For detailed information, please refer to PtWindow in the Photon documentation.

Instance Variables

`cursor_override`

A number specifying whether or not the cursor of this window overrides the cursors of its children. Default, 0, means no. Any other value means yes.

`default_action`

This variable is not yet documented in detail.

`force_front`

This variable is not yet documented in detail.

`help_root`

A string that comprises the root path for this window's help topic. Default is `""`.

`icon_window`

A `PtWidget` for the icon of the window. If set to `nil`, the window will use a child `PtIcon`, otherwise a default icon.

`managed_flags`

This instance variable allows for Window Manager interactions, and may be a combination of zero or more of the following flags:

Constant	Description
`Ph_WM_CLOSE`	Allows the window manager to close the window.
`Ph_WM_FOCUS`	Allows the window manager to handle focus on the window.
`Ph_WM_MENU`	Allows the window to generate a 'window' menu.
`Ph_WM_TOFRONT`	Sends the window to the front.
`Ph_WM_TOBACK`	Sends the window to the back.
`Ph_WM_HIDE`	Allows the window to be hidden.
`Ph_WM_CONSWITCH`	Moves the window to keep it visible when the virtual console is switched.
`Ph_WM_RESIZE`	Allows the window to be resized.
`Ph_WM_MOVE`	Allows the window to be moved.
`Ph_WM_ICON`	Iconifies the window.
`Ph_WM_MAX`	Maximizes the window.
`Ph_WM_BACKDROP`	Makes the window into the backdrop.
`Ph_WM_HELP`	Allows context-sensitive help.
`Ph_WM_RESTORE`	De-iconifies the window, or un-maximizes the window.
`Ph_WM_FFRONT`	Forces the window to the front.

`max_height`
`max_width`

A number of pixels that specify the maximum height or width that the window can be sized to.

`min_height`
`min_width`

A number of pixels that specify the minimum height or width that the window can be sized to. Defaults are 43 and 71 respectively.

`notify_flags`

Flags that allow the window to control some of its functioning locally (without help from the Window Manager). Default is `Ph_WM_RESIZE | Ph_WM_CLOSE | Ph_WM_HELP`.

This instance variable may be a combination of zero or more of the following flags:

Constant	Description
`Ph_WM_CLOSE`	Tells the program when a window close has been attempted.
`Ph_WM_FOCUS`	Tells the program when the window gains or loses focus.
`Ph_WM_MENU`	Tells the program when the 'window' menu has been selected.

Constant	Description
Ph_WM_TOFRONT	Tells the program when the window is sent to the front.
Ph_WM_TOBACK	Tells the program when the window is sent to the back.
Ph_WM_HIDE	Tells the program when the window has been hidden.
Ph_WM_CONSWITCH	Tells the program when the window has switched virtual consoles.
Ph_WM_RESIZE	Tells the program when the window has been resized.
Ph_WM_MOVE	Tells the program when the window has been moved.
Ph_WM_ICON	Tells the program when the window has been iconified.
Ph_WM_MAX	Tells the program when the window has been maximized.
Ph_WM_BACKDROP	Tells the program when the window has been made the backdrop.
Ph_WM_HELP	Tells the program when context sensitive help has been selected.
Ph_WM_RESTORE	Tells the program when context sensitive help has been restored.
Ph_WM_FFRONT	Tells the program when context sensitive help has been forced to the front.

render_flags

Flags that control how the window is rendered. Default is Ph_WM_APP_DEF_RENDER, which is a combination of several of these flags.

This instance variable may be a combination of zero or more of the following flags:

Constant	Description
Ph_WM_RENDER_ASAPP	Renders the window as an application window.
Ph_WM_RENDER_ASICON	Renders the window as an icon window.
Ph_WM_RENDER_ASMASK	As Mask
Ph_WM_RENDER_TITLE	Draws a title bar on the window.
Ph_WM_RENDER_CLOSE	Draws a close button in the window title bar.
Ph_WM_RENDER_HELP	Draws a help button in the window title bar.
Ph_WM_RENDER_MAX	Draws a maximize button in the window title bar.
Ph_WM_RENDER_MENU	Draws a menu button in the window title bar.
Ph_WM_RENDER_MIN	Draws a minimize button in the window title bar.
Ph_WM_RENDER_BORDER	Draws borders around the window.
Ph_WM_RENDER_RESIZE	Draws resizing borders around the window.

`state`

This instance variable specifies the state of the window when it is first opened, and may have one of the following values:

Constant	Description
`Ph_WM_STATE_ISMASK`	Internal informational bit.
`Ph_WM_STATE_ISHIDDEN`	The window opens normally, but is not displayed.
`Ph_WM_STATE_ISMAX`	Opens the window to its maximum size.
`Ph_WM_STATE_ISBACKDROP`	Opens the window as a workspace backdrop.
`Ph_WM_STATE_ISTASKBAR`	Internal informational bit.
`Ph_WM_STATE_ISPDM`	Internal informational bit.
`Ph_WM_STATE_ISICONIFIED`	Opens the window and makes it an icon.
`Ph_WM_STATE_ISICON`	Internal informational bit.
`Ph_WM_STATE_ISFRONT`	Opens the window in front of all other applications.
`Ph_WM_STATE_ISFOCUS`	Opens the window as the focus widget, if the WM cursor focus option is disabled. This is the default.

`title`

A string comprising the title of the window, which appears in the title bar. Default is `"Untitled"`.

`title_color`

A number specifying the color of the title bar. Default is `Pg_DEFAULT_WM_COLOR`.

`window_active_color`

A number specifying the color of the frame of the window when the window has focus. Default is `Pg_DEFAULT_WM_COLOR`.

`window_inactive_color`

A number specifying the color of the frame of the window when the window doesn't have focus. Default is `Pg_DEFAULT_WM_COLOR`.

Callbacks

The following callbacks are associated with this widget:

Callback	Description
`Pt_CB_WINDOW`	This callback is generated when an event is received from the Window Manager.
`Pt_CB_WINDOW_CLOSING`	This callback is generated as the window closes, before the window region is removed.
`Pt_CB_WINDOW_OPENING`	This callback is generated as the window is being opened, useful for resizing and anchoring.

Callback	Description
`Pt_CB_WINDOW_TRANSPORT`	This callback is generated when a window is transported through a Jump Gate.

Convenience Functions

Arguments

`widget`	A PtWindow widget.

Functions

These functions are extensions of QNX Photon functions. You can refer to the PtWindow function documentation in QNX Helpviewer for more information about them.

`PtWindowtoBack` `(widget)` -- moves the window *`widget`* and all of its child windows to the back of the workspace, but not behind its parent window. To enable a child window to be moved behind its parent, make the two windows siblings by setting the child's parent to `nil`.

Returns `t`.

`PtWindowtoFront` `(widget)` -- moves the window *`widget`* and all of its child windows to the front of the workspace, and gives it focus. A parent window cannot be moved in front of its child, however. To do this, make the windows siblings by setting the child's parent to `nil`.

Returns `t`.

RtMeter

RtMeter — An arc-shaped real-time meter with indicator needle (in QNX 4).

Synopsis

```
class RtMeter PtBasic
{
    meter_color;                // color  (Rt_ARG_METER_COLOR)
    meter_flags;                // flag  (Rt_ARG_METER_FLAGS)
    meter_font_color;           // color  (Rt_ARG_METER_FONT_COLOR)
    meter_increment;            // integer  (Rt_ARG_METER_INCREMENT)
    meter_key_left;             // integer  (Rt_ARG_METER_KEY_LEFT)
    meter_key_right;            // integer  (Rt_ARG_METER_KEY_RIGHT)
    meter_level1_color;         // color  (Rt_ARG_METER_LEVEL1_COLOR)
    meter_level1_pos;           // short  (Rt_ARG_METER_LEVEL1_POS)
    meter_level2_color;         // color  (Rt_ARG_METER_LEVEL2_COLOR)
    meter_level2_pos;           // short  (Rt_ARG_METER_LEVEL2_POS)
    meter_level3_color;         // color  (Rt_ARG_METER_LEVEL3_COLOR)
    meter_max_needle_position;  // short  (Rt_ARG_METER_MAX_NEEDLE_POSITION)
    meter_min_needle_position;  // short  (Rt_ARG_METER_MIN_NEEDLE_POSITION)
    meter_needle_color;         // color  (Rt_ARG_METER_NEEDLE_COLOR)
    meter_needle_position;      // short  (Rt_ARG_METER_NEEDLE_POSITION)
    meter_num_severity_levels;  // short  (Rt_ARG_METER_NUM_SEVERITY_LEVELS)
    meter_text_font;            // string  (Rt_ARG_METER_TEXT_FONT)
}
```

Base Classes

PtWidget <-- PtBasic <-- RtMeter

Description

This widget is a half-circle meter with an indicator needle and up to three different-colored pie-shaped arcs that mark three chosen ranges on the meter. The QNX 6 version of this widget is PtMeter.

For detailed information, please refer to RtMeter in the Photon documentation.

Instance Variables

meter_color

A number specifying the color used for the meter's center, outline, and tick marks. Default is 0x0 (black).

meter_flags

This instance variable controls characteristics of the widget, and may have one or two of the following values:

Constant	Description
RtM_NON_SELECTABLE	The meter is non-interactive.
RtM_SELECTABLE	The meter is interactive. Default is ON.
RtM_NO_TEXT	The minimum and maximum values are not displayed.

`meter_font_color`

A number specifying the color for the display of minimum and maximum values. Default is `0x0` (black).

`meter_increment`

A number of units to move the needle when an assigned key is pressed.

`meter_key_left`
`meter_key_right`

A number or key name from `PkKeyDef.lsp` for the key that moves the needle to the left or right. Defaults are Pk_Left and Pk_Right. You can load the key name constants with a call to **`require_lisp("PkKeyDef.lsp")`**.

`meter_level1_color`

A number specifying a color for the left arc of the meter that corresponds to Level 1. Default is `0x00ff00` (green).

`meter_level1_pos`

A number specifying the end of Level 1, expressed as a percentage of the whole meter. Default is 50. This setting is unaffected by changes to `meter_max_needle_position` or `meter_min_needle_position`.

`meter_level2_color`

A number specifying a color for the center arc of the meter that corresponds to Level 2. Default is `0xffff00` (yellow).

`meter_level2_pos`

A number specifying the end of Level 2, expressed as a percentage of the whole meter. Default is 75. This setting is unaffected by changes to `meter_max_needle_position` or `meter_min_needle_position`.

`meter_level3_color`

A number specifying a color for the right arc of the meter that corresponds to Level 3. Default is `0xff0000` (red).

`meter_max_needle_position`

A number specifying the maximum point on the meter, the maximum value for the needle to register.

`meter_min_needle_position`

A number specifying the minimum point on the meter, the minimum value for the needle to register.

`meter_needle_color`

A number specifying the color of the needle. Default is `0xffffff` (white).

`meter_needle_position`

A number specifying the current position of the needle, within the range of `meter_min_needle_position` and `meter_max_needle_position`. The needle will remain at the maximum or minimum position for any value outside that range.

`meter_num_severity_levels`

The number of arcs that the meter is divided into. The maximum number is 3, which is also the default.

`meter_text_font`

A string specifying the font used for displaying the maximum and minimum values. Default is `"helv10"`.

Associated Classes

`RtMeterCallback`, `PtCallbackInfo`

Example

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

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

/*
 * This example demonstrates an RtMeter in QNX 4, or a
 * PtMeter in QNX 6.
 */

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

function display_value(wnum, wmet)
{
  wmet.meter_needle_position = wnum.numeric_value;
}

function reset_num(wnum, wmet)
{
  wnum.numeric_value = wmet.meter_needle_position;
}

win = new(PtWindow);
win.SetArea(100, 100, 250, 250);
if (_os_ == "QNX4")
     meter = new(RtMeter);
else if (_os_ == "QNX6")
     meter = new(PtMeter);

meter.SetArea(20, 40, 200, 200);
meter.meter_color = 0xccddff;
meter.meter_max_needle_position = 150;
meter.meter_min_needle_position = 0;
meter.meter_level1_color = 0x00ff00;
meter.meter_level1_pos = 50;
meter.meter_level2_color = 0xffff00;
meter.meter_level2_pos = 100;
meter.meter_level3_color = 0xff0000;
meter.meter_needle_position = 70;
meter.meter_needle_color = 0x000000;

num = new(PtNumericInteger);
num.SetArea(100, 150, 50, 20);
num.numeric_increment = 10;
num.numeric_min = 0;
num.numeric_max = 150;
num.numeric_value = 70;
num.numeric_flags = cons(Pt_NUMERIC_WRAP, nil);
num.numeric_flags = cons(Pt_NUMERIC_AUTO_HIGHLIGHT, nil);

label = new(PtLabel);
label.SetPos(10, 190);
label.text_font = "helv09";
label.text_string = string("Adjust the meter by clicking anywhere,\n",
      "by dragging the needle, or by changing \n",
```

```
      "the value in the PtNumeric widget.");

if (_os_ == "QNX4")
     PtAttachCallback(meter, Rt_CB_METER_MOVED, ‘reset_num(@num, @meter));
else if (_os_ == "QNX6")
     PtAttachCallback(meter, Pt_CB_METER_MOVED, ‘reset_num(@num, @meter));

PtAttachCallback(num, Pt_CB_NUMERIC_CHANGED, ‘display_value(@num, @meter));
display_value(num, meter);

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

RtProgress

`RtProgress` — A bar-style real-time progress indicator.

Synopsis

```
class RtProgress PtGauge
{
    progress_bar_color;    // color  (Rt_ARG_PROGRESS_BAR_COLOR)
    progress_divisions;    // unsigned short  (Rt_ARG_PROGRESS_DIVISIONS)
    progress_gap;          // unsigned short  (Rt_ARG_PROGRESS_GAP)
    progress_spacing;      // unsigned short  (Rt_ARG_PROGRESS_SPACING)
}
```

Base Classes

`PtWidget` <-- `PtBasic` <-- `PtGauge` <-- `RtProgress`

Description

This widget is a bar that fills with color from left to right to indicate progress from 0% to 100%. The colors can be changed, and the progress bar can be divided, with spaces put between the divisions.

For detailed information, please refer to RtProgress in the Photon documentation.

Instance Variables

`progress_bar_color`

A number specifying the color of the progress bar. Default is `0xff0000` (red).

`progress_divisions`

A number specifying how many parts the bar is divided into. Default is `1`. If divided into more than one part, the bar will not register a change in value until the entire part is complete. For example, a bar of 100 units divided into 10 parts whose value is 37 will show progress of 30 units. When the value reaches 40, the bar will show 40 units.

`progress_gap`

This variable is not currently implemented.

`progress_spacing`

A number of pixels specifying the separation between the parts, as divided by the `progress_divisions` variable above. The coloring of these spaces is the same as is set in the inherited `fill_color` variable

Example

This example, `ex_PtNumeric-RtProgress.g`, is included in the product distribution.

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

/*
 * This example demonstrates PtNumeric and
 * PtProgress widgets.
 */

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

```
function display_value(wnum, wprog)
{
  wprog.gauge_value = wnum.numeric_value;
}

win = new(PtWindow);
win.SetDim(160, 190);

ilab = new(PtLabel);
ilab.text_string = "Integer values";
ilab.SetArea(20, 6, 120, 20);
ilab.horizontal_alignment = Pt_CENTER;

inum = new(PtNumericInteger);
inum.SetArea(20, 30, 120, 20);
inum.numeric_increment = 5;
inum.numeric_max = 50;
inum.numeric_min = -50;
inum.numeric_value = 5;
inum.numeric_flags = cons(Pt_NUMERIC_WRAP, nil);
inum.numeric_flags = cons(Pt_NUMERIC_AUTO_HIGHLIGHT, nil);

if (_os_ == "QNX4")
     iprog = new(RtProgress);
else
     iprog = new(PtProgress);
iprog.SetArea(20, 60, 115, 20);
iprog.gauge_minimum = -50;
iprog.gauge_maximum = 50;

PtAttachCallback(inum, Pt_CB_NUMERIC_CHANGED, ‘display_value(@inum, @iprog));
display_value(inum, iprog);

if (_os_ == "QNX4")
{
  flab = new(PtLabel);
  flab.text_string = "Float values";
  flab.SetArea(20, 96, 120, 20);
  flab.horizontal_alignment = Pt_CENTER;

  fnum = new(PtNumericFloat);
  fnum.SetArea(20, 120, 120, 20);
  fnum.numeric_increment = 4.8941;
  fnum.numeric_max = 50;
  fnum.numeric_min = -50;
  fnum.numeric_value = 5.34153;
  fnum.numeric_precision = 4;
  fnum.numeric_flags = cons(Pt_NUMERIC_WRAP, nil);
  fnum.numeric_flags = cons(Pt_NUMERIC_AUTO_HIGHLIGHT, nil);

  fprog = new(RtProgress);
  fprog.SetArea(20, 150, 115, 20);
  fprog.gauge_minimum = -50;
  fprog.gauge_maximum = 50;
  fprog.progress_bar_color = 0x0000ff;
  PtAttachCallback(fnum, Pt_CB_NUMERIC_CHANGED, ‘display_value(@fnum, @fprog));
  display_value(fnum, fprog);
}

else
{
  lab = new(PtLabel);
  lab.SetPos(15, 90);
  lab.text_font  = "helv8";
  lab.text_string = string("Note: This example in QNX 4\n",
      "also contains a PtNumeric Float \n",
      "widget, which is not currently\n",
      "available in Gamma/Photon for\n",
```

```
      "QNX 6.");
}

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

RtTrend

RtTrend — A graph for displaying real-time trends (in QNX 4).

Synopsis

```
class RtTrend PtBasic
{
    rttrend_data;        // array of arrays of integer  (Rt_ARG_TREND_DATA)
    rttrend_flags;       // flag  (Rt_ARG_TREND_FLAGS)
    trend_attributes;    // integer array  (Rt_ARG_TREND_ATTRIBUTES)
    trend_color_list;    // color array  (Rt_ARG_TREND_COLOR_LIST)
    trend_count;         // integer  (Rt_ARG_TREND_COUNT)
    trend_grid_color;    // color  (Rt_ARG_TREND_GRID_COLOR)
    trend_grid_x;        // short  (Rt_ARG_TREND_GRID_X)
    trend_grid_y;        // short  (Rt_ARG_TREND_GRID_Y)
    trend_inc;           // short  (Rt_ARG_TREND_INC)
    trend_max;           // short  (Rt_ARG_TREND_MAX)
    trend_min;           // short  (Rt_ARG_TREND_MIN)
}
```

Base Classes

PtWidget <-- PtBasic <-- RtTrend

Description

This widget is a real-time trend display that can plot multiple trends and display them in a moving format, to show changes over time. The QNX 6 version of this widget is PtTrend.

For detailed information, please refer to RtTrend in the Photon documentation.

Instance Variables

rttrend_data

An array of data arrays. Each data array corresponds to one trend line on the display.

rttrend_flags

Flags that specify grid characteristics and direction. If any of these flags is set incorrectly, the widget will not display any trends.

The grid options are only supported by 8-bit graphics environments, and only on some graphics cards.

This instance variable may be a combination of zero or more of the following flags:

Constant	Description
Rt_GRID	Draws a grid. This flag requires exactly one of Rt_GRID_IS_TRANSLUCENT, Rt_GRID_ABOVE_TRENDS, or Rt_TRENDS_ABOVE_GRID to be set as well.
Rt_GRID_IS_TRANSLUCENT	Makes the grid that was set using Rt_GRID appear to be translucent.
Rt_GRID_ABOVE_TRENDS	Makes the grid that was set using Rt_GRID appear to be superimposed over the trends.

Constant	Description
Rt_GRID_FORCE	Forces drawing the grid, despite possible flickering effect due to poor hardware support.
Rt_PIXEL	The grid is drawn using a pixel array instead of vector graphics.
Rt_TRENDS_ABOVE_GRID	Makes the grid that was set using Rt_GRID appear to be underneath the trends.
Rt_TREND_HORIZONTAL	Causes the trend to move horizontally. This flag requires exactly one of Rt_TREND_LEFT_TO_RIGHT or Rt_TREND_RIGHT_TO_LEFT.
Rt_TREND_VERTICAL	Causes the trend to move vertically. This flag requires exactly one of Rt_TREND_TOP_TO_BOTTOM or Rt_TREND_BOTTOM_TO_TOP.
Rt_LEFT_TO_RIGHT	Sets the direction for Rt_TREND_HORIZONTAL.
Rt_RIGHT_TO_LEFT	Sets the direction for Rt_TREND_HORIZONTAL.
Rt_TOP_TO_BOTTOM	Sets the direction for Rt_TREND_VERTICAL.
Rt_BOTTOM_TO_TOP	Sets the direction for Rt_TREND_VERTICAL.

`trend_attributes`

An array that indexes the `trend_color_list`. Index numbers are consecutive integers starting with 0, and each number points to an element of the `trend_color_list`. Default is `nil`.

`trend_color_list`

An array of colors to be used for plotting trend lines. Default array has a single number: [16711680], which equals 0xff0000 (red).

`trend_count`

An integer specifying the number of trends.

`trend_grid_color`

A number specifying the color of the grid to be used if the `Rt_GRID` flag is set in the `rttrend_flags` variable. Default is `0xc0c0c0` (grey).

`trend_grid_x`

A number specifying the number of vertical grid lines to be used if the `Rt_GRID` flag is set in the `rttrend_flags` variable. Default is `5`.

`trend_grid_y`

A number specifying the number of horizontal grid lines to be used if the `Rt_GRID` flag is set in the `rttrend_flags` variable. Default is `5`.

`trend_inc`

A number specifying the spacing of the plotted points along the moving axis. Default is `1`.

`trend_max`

A number specifying the maximum value for the display. Default is `32,767`.

`trend_min`

A number specifying the maximum value for the display. Default is `-32,768`.

Example

Here is a very simple example of an `RtTrend` widget. For more complex examples, see the Gamma demos **`cpumem`** and **`trend`**. This example, `ex_Trend.g`, is included in the product distribution.

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

/*
 * This example illustrates an RtTrend in QNX 4, or a PtTrend in QNX 6,
 * by creating and displaying a sine function.
 */
require_lisp("PhotonWidgets.lsp");
require_lisp("PhabTemplate.lsp");
PtInit(nil);

wfile = PhabReadWidgetFile (string(_os_, "-WidgetFiles/wgt/trend.wgtw"));
window = PhabCreateWidgets(wfile, nil, nil);
win = PhabLookupWidget(window,#trend,nil);
tr = PhabLookupWidget(window,#Trend,nil);
ebut = PhabLookupWidget(window,#TrendExitButton,nil);
PtAttachCallback(ebut, Pt_CB_ACTIVATE, #exit_program(1));

Values := make_array (1);
valarray = make_array (1);

/*
 * Make a function to generate a trend.
 */

Sinex = 0;

function Sine (inc, max, min)
{
 local  temp = sin (Sinex), diff = (max - min) / 2;
 Sinex = Sinex + inc;
 temp * diff + min + diff;
}

Trend = list (Sine, 0.07, 550, 250);

/*
 * Make a function to evaluate the trend.
 */

function FillTrend ()
{
  valarray[0] = eval(Trend);
  Values[0] = valarray;
  if (_os_ == "QNX4")
    tr.rttrend_data = Values;
  else if (_os_ == "QNX6")
    tr.trend_data = Values;
}

/*
 * Set up the trend.
 */

tr.trend_max = 800;
tr.trend_min = 0;

/*
 * Start an update timer, and loop forever.
 */
```

```
every (.03, #FillTrend());

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

II. Callback Classes

Table of Contents

CwMatrixCallback

`CwMatrixCallback` — matrix activity callback information.

Synopsis

```
class CwMatrixCallback
{
    cell;           // CwMatrixCell
    cur_range;      // PhRect
    dim;            // PhDim
    flags;          // long
    prev_range;     // PhRect
    prev_string;    // string
    text_string;    // string
}
```

Description

This class holds information from the Cw_CB_MATRIX_BEGIN_EDIT, Cw_CB_MATRIX_CELL_CHANGE, Cw_CB_MATRIX_CHARINPUT, Cw_CB_MATRIX_KEYPRESS, Cw_CB_MATRIX_RANGE_CONVERT, Cw_CB_MATRIX_RANGE_MOVE, Cw_CB_MATRIX_RANGE_SEL, Cw_CB_MATRIX_SIZE_CHANGE, and Cw_CB_MATRIX_TEXT_CHANGE callbacks, which are generated by activity on a `CwMatrix`.

Instance Variables

`cell`

The `CwMatrixCell` related to the callback.

`cur_range`

A `PhRect` indicating the current range.

`dim`

A `PhDim`.

`flags`

This instance variable may be a combination of zero or more of the following flags:

Constant	Description
`Pt_HIGHLIGHTED`	Draw a beveled border around the widget.
`Pt_AUTOHIGHLIGHT`	Automatically display/remove the highlight border as the cursor passes over the widget.
`Pt_ETCH_HIGHLIGHT`	Draw a double bevel if `Pt_HIGHLIGHTED` is on.
`Pt_SET`	Make the widget 'set'. This will invert the coloring on the bevel as well.
`Pt_TOGGLE`	The widget's SET flag will toggle on each mouse click instead of changing with mouse down/mouse up events.
`Pt_SELECTABLE`	The widget may be selected, causing `Pt_CB_ARM/ACTIVATE/DISARM` events.
`Pt_GHOST`	The widget is displayed 'ghosted'. This does not affect its response to events.

Constant	Description
Pt_BLOCKED	The widget will be unresponsive to events.
Pt_REALIZED	The widget is visible on the display.
Pt_CLIP_HIGHLIGHT	The corners of the highlight rectangle are clipped off.
Pt_OPAQUE	Makes the widget opaque within its own extent and for everything behind it.
Pt_DELAY_REALIZE	The widget won't be realized except by a call to PtRealizeWidget().
Pt_GETS_FOCUS	The widget may get the keyboard focus.
Pt_MENU_BUTTON	Makes the widget a menu item.
Pt_DESTROYED	Marks the widget for destruction.
Pt_DAMAGED	Marks the widget for repairs.
Pt_OBSCURED	The widget is covered by another widget, or is outside its parent's canvas.
Pt_IN_FLUX	A call to PtContainerHold() has been made on the widget.
Pt_CLEAR	Keeps the widget's extent clear of any brothers in front of it.
Pt_DAMAGE_FAMILY	The widget and its children will be repaired.
Pt_SELECT_NOREDRAW	The widget will not redraw when it is selected.
Pt_WIDGET_REBUILD	The widget will be rebuilt once all resources have been changed.
Pt_WIDGET_RESIZE	The widget will be resized once all resources have been changed.
Pt_PROCREATED	The widget is a procreated child of a compound widget.
Pt_ALL_BUTTONS	The widget treats events on any mouse button as a selection.
Pt_FOCUS_RENDER	The widget will attempt to indicate that it has the keyboard focus through some means.
Pt_CALLBACKS_ACTIVE	Callbacks for this widget will be called due to changes through code, not just due to user interactions.
Pt_MENUABLE	This widget will respond to the menu button with a Pt_CB_MENU event.
Pt_NOREDRAW_SET	Noredraw Set
Pt_FREE_MEMORY	Frees memory associated with widget pointers.
Pt_REGION	Force the widget to have a region ID.
Pt_REALIZING	The widget is being realized.

`prev_range`

A `PhRect` indicating the previously selected range.

`prev_string`

A string indicating the previous text string content of the cell.

`text_string`

A string indicating the current text string content of the cell.

Associated Classes

`CwMatrix`, `CwMatrixCell`, `PtCallbackInfo`

PtBasicCallback

PtBasicCallback — toggle status callback information.

Synopsis

```
class PtBasicCallback
{
    value;    // integer
}
```

Description

This class holds information from a Pt_CB_ACTIVATE callback.

For detailed information, please refer to Pt_CB_ACTIVATE in the PtBasic section of the Photon documentation.

Instance Variables

value

A number indicating the toggle status for button-style widgets (1 for IN, 0 for OUT), or the click count if Pt_TOGGLE isn't set.

Associated Classes

PtBasic, PtCallbackInfo

PtCalendarSelectCallback

`PtCalendarSelectCallback` — calendar selection callback information.

Synopsis

```
class PtCalendarSelectCallback
{
    date;    // PtCalendarDate
    time;    // long
    type;    // integer
}
```

Description

This class holds information from the `Pt_CB_CALENDAR_SELECT` callback, which is generated when a calendar date, weekday, month increment/decrement button or year increment/decrement button is selected in a PtCalendar.

Also see Pt_CB_CALENDAR_SELECT in the PtCalendar section of the Photon documentation.

Instance Variables

`date`

A `PtCalendarDate` that contains the day, month, and year.

`time`

The system time associated with the `date`.

`type`

The type of selection made, which can be exactly one of the following:

Constant	Description
`Pt_CALENDAR_DATE_SELECTED`	A date was selected. Numerical value = 1.
`Pt_CALENDAR_WDAY_SELECTED`	A day of the week was selected. Numerical value = 2.
`Pt_CALENDAR_MONTH_SELECTED`	One of the month increment/decrement buttons was selected. Numerical value = 3.
`Pt_CALENDAR_YEAR_SELECTED`	One of the year increment/decrement buttons was selected. Numerical value = 4.

Associated Classes

`PtCalendar`, `PtCallbackInfo`

PtCallbackInfo

`PtCallbackInfo` — gives access to callback information.

Synopsis

```
class PtCallbackInfo
{
    event;                      // PhEvent
    reason;                     // unsigned long
    reason_subtype;             // unsigned long

// one from the following group, depending on the widget:

    basic;                      // PtBasicCallback
    calendar;                   // PtCalendarSelectCallback
    clock_time;                 // PtClockTimeCallback
    container;                  // PtContainerCallback
    divider;                    // PtDividerCallback
    filesel;                    // PtFileSelCallback
    filesel_bkgd;               // PtFileSelBkgdCallback
    font_name;                  // string
    gen_tree_input;             // PtGenTreeInput
    graphic;                    // PhArea
    hotkey;                     // PtHotkeyCallback
    html;                       // PtHtmlCallback
    list;                       // PtListCallback
    list_input;                 // PtListInput
    matrix;                     // CwMatrixCallback
    numeric_float;              // PtNumericFloatCallback
    numeric_integer;            // PtNumericIntegerCallback
    on_off_button;              // PtOnOffButtonCallback
    rtmeter;                    // RtMeterCallback
    scrollbar;                  // PtScrollbarCallback
    slider;                     // PtSliderCallback
    terminal_font_change;       // PtTerminalFontChange
    terminal_input;             // PtTerminalInput
    terminal_option_change;     // PtTerminalOptionChange
    terminal_scrlbk_cb;         // PtTerminalScrlbkCb
    terminal_size_change;       // PtTerminalSizeChange
    text;                       // PtTextCallback
    tree;                       // PtTreeCallback
    tty_output;                 // PtTtyOutput
}
```

Description

This class gives access to callback information. It has three normal instance variables, and a group of widget-specific instance variables that correspond to one or more widgets in Gamma.

Whenever a callback is generated, a locally-scoped instance of `PtCallbackInfo`, called `cbinfo`, is created. It has the three regular instance variables, as well as one instance variable from the widget-specific group. You can access these callback variables using dot notation.

Only one of the instance variables from the group of widget-specific instance variables should be used: the one that actually corresponds to the widget. Using an instance variable that doesn't correspond to the widget will give unpredictable results.

Instance Variables

`event`

A `PhEvent` that caused the generation of the callback.

`reason`

The value of the callback that was generated.

`reason_subtype`

The value corresponding to the way the callback was generated, if more than one way was possible. The possibilities are callback-specific, and are documented in detail with the relevant widget in the Photon documentation.

Widget-Specific Instance Variables

Most of the instance variables from the widget-specific group have a callback class associated with them, and they are documented with that class. Those that don't are documented here.

`font_name`

A string comprising the name of the new font. This information comes from the `Pt_CB_FONT_MODIFY` callback, generated by `PtFontSel`. Also see Pt_CB_FONT_MODIFY in the PtFontSel section of the Photon documentation.

`graphic`

A `PhArea` indicating the previous dimensions of the graphic. This information comes from the `Pt_CB_RESCALE` callback, which is generated when a PtGraphic is rescaled. Also see Pt_CB_RESCALE in the PtGraphic section of the Photon documentation.

Examples

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

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


/*
 * This example demonstrates how to access raw callback information.
 */


Ph_EV_PTR_STEADY := 2;
Ph_EV_PTR_UNSTEADY := 3;

function main ()
{
  require_lisp("PhotonWidgets.lsp");
  PtInit (nil);

  w = new (PtWindow);
  w.SetArea(100, 100, 120, 80);
  b = new (PtButton);
  b.SetPos(30, 30);
  b.text_string = "Test Area";

  PtAttachCallback (b, Pt_CB_RAW, #cbBoundary(), Ph_EV_BOUNDARY);
  PtRealizeWidget (w);

  PtMainLoop();
}

/* widget, cbinfo, event_data are all available here */
function cbBoundary ()
{
 local  event;

 event = cbinfo.event;
```

```
 if (event.subtype == Ph_EV_PTR_ENTER)
  princ ("Enter\n");
 else if (event.subtype == Ph_EV_PTR_LEAVE)
  princ ("Leave\n");
 else if (event.subtype == Ph_EV_PTR_STEADY)
  princ ("Steady\n");
 else if (event.subtype == Ph_EV_PTR_UNSTEADY)
  princ ("Unsteady\n");
 else
  princ ("unexpected event subtype\n");
}
```

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

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

/*
 * This example demonstrates how to get information from a callback,
 * the PtCalendarSelectCallback in this case.  The first three
 * callback functions print the regular instance variables.  The next
 * callback function prints the PtCalendarSelectCallback instance
 * variable named date.  Then the instance variables of date are
 * printed by the next three callback functions. The last two
 * callback functions print the remaining two variables of
 * PtCalendarSelect.
 */

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

win = new(PtWindow);
win.SetPos (300,0);

cal = new(PtCalendar);
cal.SetDim(200,200);
cal.fill_color = 0x88ddbb;

PtAttachCallback(cal,Pt_CB_CALENDAR_SELECT,
                 #pretty_princ("Event: ",cbinfo.event,"\n"));
PtAttachCallback(cal,Pt_CB_CALENDAR_SELECT,
                 #princ("Reason: ",cbinfo.reason,"\n"));
PtAttachCallback(cal,Pt_CB_CALENDAR_SELECT,
                 #princ("Subtype: ",cbinfo.reason_subtype,"\n"));
PtAttachCallback(cal,Pt_CB_CALENDAR_SELECT,
                 #princ("Day: ",cbinfo.calendar.date.day + 1,"\n"));
PtAttachCallback(cal,Pt_CB_CALENDAR_SELECT,
                 #princ("Month: ",cbinfo.calendar.date.month + 1,"\n"));
PtAttachCallback(cal,Pt_CB_CALENDAR_SELECT,
                 #princ("Year: ",cbinfo.calendar.date.year,"\n"));
PtAttachCallback(cal,Pt_CB_CALENDAR_SELECT,
                 #princ("Time: ",cbinfo.calendar.time,"\n"));
PtAttachCallback(cal,Pt_CB_CALENDAR_SELECT,
                #princ("Type: ",cbinfo.calendar.type,"\n\n"));

princ(Pt_CB_CALENDAR_SELECT);

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

See Also

PtAttachCallback

PtClockTimeCallback

PtClockTimeCallback — time change callback information.

Synopsis

```
class PtClockTimeCallback
{
    hour;          // short
    minute;        // short
    old_hour;      // short
    old_minute;    // short
    old_second;    // short
    second;        // short
}
```

Description

This class holds information from the Pt_CB_CLOCK_TIME_CHANGED callback, which is generated when the time changes on a PtClock.

Also see Pt_CB_CLOCK_TIME_CHANGED in the PtClock section of the Photon documentation.

Instance Variables

hour

A number indicating the clock's new hour setting, in 24 hour format, and including any hour offset.

minute

A number indicating the clock's new minute setting.

old_hour

A number indicating the clock's previous hour setting, in 24 hour format, and including any hour offset.

old_minute

A number indicating the clock's previous minute setting.

old_second

A number indicating the clock's previous second setting.

second

A number indicating the clock's new second setting.

Associated Classes

PtClock, PtCallbackInfo

PtContainerCallback

PtContainerCallback — resize callback information.

Synopsis

```
class PtContainerCallback
{
    new_size;    // PtRect
    old_size;    // PtRect
}
```

Description

This class holds information from the Pt_CB_RESIZE callback, which is generated when the container is resized.

Also see Pt_CB_RESIZE in the PtContainer section of the Photon documentation.

Instance Variables

new_size

A PtRect defining the new size of the container.

old_size

A PtRect defining the former size of the container.

Associated Classes

PtContainer, PtCallbackInfo

PtDividerCallback

`PtDividerCallback` — divider movement callback information.

Synopsis

```
class PtDividerCallback
{
    left;       // widget array
    moved;      // integer
    resized;    // integer
    right;      // widget array
}
```

Description

This class holds information from a `Pt_CB_DIVIDER_DRAG` callback, which is generated when a `PtDivider` is moved.

Also see Pt_CB_DIVIDER_DRAG in the PtDivider section of the Photon documentation.

Instance Variables

`left`

The widget on the left side of the divider.

`moved`

An integer specifying the number of pixels that the widgets on the right of the divider were moved.

`resized`

An integer specifying the number of pixels that the widget on the left of the divider was resized.

`right`

The widget on the right side of the divider.

Associated Classes

`PtDivider`, `PtCallbackInfo`

PtFileSelCallback

PtFileSelCallback — file selection callback information.

Synopsis

```
class PtFileSelCallback
{
    item;        // PtFileSelItem
    nitems;      // unsigned integer
    reason;      // short
    sel_mode;    // unsigned integer
}
```

Description

This class holds information from a Pt_CB_FS_STATE or Pt_CB_FS_SELECTION callback. These callbacks are generated when a file is selected in a PtFileSel.

Also see Pt_CB_FS_STATE or Pt_CB_FS_SELECTION in the PtFileSel section of the Photon documentation.

Instance Variables

item

A PtFileSelItem that has been selected.

nitems

Used by Pt_CB_FS_SELECTION only; the number of items selected.

reason

Used by Pt_CB_FS_STATE only; one of the following constants is possible:

Constant	Definition
Pt_FS_STATE_START	The callback is being called before the disk is read.
Pt_FS_STATE_END	The callback is being called after the disk is read.

sel_mode

An integer specifying the selection mode, which can be one or more of the following:

Constant	Definition
Pt_BROWSE_MODE	Click with mouse or drag and release to select one item.
Pt_MULTIPLE_MODE	Select multiple items by clicking on them. A second click releases the item.
Pt_EXTENDED_MODE	**Shift** click to select a range, **Ctrl** click to select disjoint items.
Pt_SINGLE_MODE	Select one item at a time. Selecting a second item releases the first.
Pt_RANGE_MODE	Press, drag, and release mouse to select a range of items.

Associated Classes

PtFileSel, PtFileSelBkgdCallback, PtCallbackInfo

PtFileSelBkgdCallback

`PtFileSelBkgdCallback` — directory item read callback information.

Synopsis

```
class PtFileSelBkgdCallback
{
    name;    // string
}
```

Description

This class holds information from the `Pt_CB_FS_BKGD_HANDLER` callback, which is generated when a directory item is read.

Also see Pt_CB_FS_BKGD_HANDLER in the PtFileSel section of the Photon documentation.

Instance Variables

`name`

A string indicating the item added to the widget.

Associated Classes

`PtFileSel`, `PtFileSelCallback`, `PtCallbackInfo`

PtGenTreeInput

PtGenTreeInput — input callback information.

Synopsis

```
class PtGenTreeInput
{
    consumed;    // integer
    item;        // string
    index;       // unsigned short
    pos;         // PhPoint
}
```

Description

This class holds information from the Pt_CB_GEN_TREE_INPUT callback, which is generated for any mouse or key event on a PtGenTree.

Also see Pt_CB_GEN_TREE_INPUT in the PtGenTree section of the Photon documentation.

Instance Variables

consumed

An integer used to suppress event handling, except mouse events, which are always consumed. Possible values include:

Constant	Description
Pt_CONTINUE	The default, it causes an event to be handled normally.
Pt_END	Causes an event to be consumed.
Pt_HALT	Passes an event to the parent widget.
Any other value	causes the event to be consumed, and suppresses further processing.

item

A string indicating the item that the mouse is pointing to, or the future current item after the widget processes an event.

index

The item index number.

pos

A PhPoint indicating the relative position of the pointer to the item.

Associated Classes

PtGenTree, PtCallbackInfo

PtHotkeyCallback

`PtHotkeyCallback` — hotkey callback information.

Synopsis

```
class PtHotkeyCallback
{
    data;          // string
    event_f;       // string
    flags;         // flag
    key_mods;      // unsigned long
    key_sym_cap;   // unsigned short
    widget;        // PtWidget
}
```

Description

This class holds information from the `Pt_CB_HOTKEY` callback, which is generated when a "hot key" assigned to the widget is pressed.

Also see Pt_CB_HOTKEY in the PtWidget section of the Photon documentation.

Instance Variables

`data`

User-defined data associated with the callback.

`event_f`

A string identifying the assigned function to call. If no function has been assigned, the value here will be `nil`, and the `Pt_CB_ACTIVATE callback` is generated.

`flags`

This instance variable may have one or more of the following values:

Constant	Description
`Pt_HOTKEY_SYM`	Causes the `key_cap_sym` variable to be interpreted as a key sym, rather than a key cap.
`Pt_HOTKEY_IGNORE_MODS`	Causes the `key_mods` variable to be ignored, allowing upper- and lowercase letters to be accepted as hot keys.

`key_mods`

The active key modifiers for this keystroke.

`key_sym_cap`

The key cap for this keystroke.

`widget`

The `PtWidget` that attached the callback.

Associated Classes

`PtWidget`, `PtCallbackInfo`

PtHtmlCallback

PtHtmlCallback — file loading callback information.

Synopsis

```
class PtHtmlCallback
{
    url;     // string
}
```

Description

This class holds information from four different PtHtml callbacks: Pt_CB_HTML_ERROR, Pt_CB_HTML_IMAGE, Pt_CB_HTML_POST_LOAD, and Pt_CB_HTML_PRE_LOAD.

Also see PtHtml in the Photon documentation.

Instance Variables

url

A string identifying the URL of a file, depending on the callback:

Callback	File
Pt_CB_HTML_ERROR	A file that couldn't be loaded.
Pt_CB_HTML_IMAGE	An image to be loaded.
Pt_CB_HTML_POST	The currently loaded file.
Pt_CB_HTML_PRE	The file to be loaded.

Associated Classes

PtHtml, PtCallbackInfo

PtListCallback

`PtListCallback` — item selection callback information.

Synopsis

```
class PtListCallback
{
    item;        // string
    item_len;    // integer
    item_pos;    // integer
    mode;        // unsigned integer
}
```

Description

These classes hold information from a `Pt_CB_SELECTION` callback, which is generated when an item is selected from a PtList or a PtComboBox list.

Also see Pt_CB_SELECTION in the PtList section of the Photon documentation.

Instance Variables

`item`

A string indicating the selected item.

`item_len`

A number of bytes indicating the length of the selected item.

`item_pos`

An integer indicating the position of the item in the list array.

`mode`

An integer specifying the selection mode, which is discussed in detail at Pt_ARG_SELECTION_MODE in the PtGenList section of the Photon documentation.

Associated Classes

`PtList`, `PtListInput`, `PtComboBox`, `PtCallbackInfo`

PtListInput

`PtListInput` — text input callback information.

Synopsis

```
class PtListInput
{
    consumed;    // integer
    item;        // string
    index;       // unsigned short
    pos;         // PhPoint
}
```

Description

This class holds information from the `Pt_CB_LIST_INPUT` callback, which is generated for any mouse of key event on a `PtList`.

Also see Pt_CB_LIST_INPUT in the PtList section of the Photon documentation.

Instance Variables

`consumed`

An integer used to suppress event handling, except mouse events, which are always consumed. Possible values include:

Constant	Description
`Pt_CONTINUE`	The default, it causes an event to be handled normally.
`Pt_END`	Causes an event to be consumed.
`Pt_HALT`	Passes an event to the parent widget.
Any other value	causes the event to be consumed, and suppresses further processing.

`item`

A string indicating the item that the mouse is pointing to, or the future current item after the widget processes an event.

`index`

The `item` index number.

`pos`

A `PhPoint` indicating the relative position of the pointer to the `item`.

Associated Classes

`PtList`, `PtListCallback`, `PtCallbackInfo`

PtMeterCallback

`PtMeterCallback` — needle movement callback information (in QNX 6).

Synopsis

```
class PtMeterCallback
{
    position;    // integer
    severity;    // integer
}
```

Description

This class holds information from the `Pt_CB_METER_MOVED` callback, which is generated when the meter needle moves. The QNX 4 version of this widget is `RtMeterCallback`.

Also see Pt_CB_METER_MOVED in the PtMeter section of the Photon documentation.

Instance Variables

`position`

A number indicating the needle position.

`severity`

A number indicating the severity arc that the needle is in.

Associated Classes

`PtMeter`, `PtCallbackInfo`

PtNumericFloatCallback

`PtNumericFloatCallback` — float change callback information.

Synopsis

```
class PtNumericFloatCallback
{
    numeric_value;    // double
}
```

Description

This class holds information from the `Pt_CB_NUMERIC_CHANGED` callback, which is generated when a `PtNumericFloat` widget changes its value.

The `Pt_CB_NUMERIC_CHANGED` callback is used by both `PtNumericFloatCallback` and `PtNumericIntegerCallback`, but the type of `numeric_value` is double or integer, depending on where the callback was used.

Also see Pt_CB_NUMERIC_CHANGED in the PtNumericFloat section of the Photon documentation.

Instance Variables

`numeric_value`

A number indicating the current value of the widget.

Associated Classes

`PtNumeric`, `PtNumericFloat`, `PtNumericIntegerCallback`, `PtCallbackInfo`

PtNumericIntegerCallback

PtNumericIntegerCallback — integer change callback information.

Synopsis

```
class PtNumericIntegerCallback
{
    numeric_value;    // integer
}
```

Description

This class holds information from the Pt_CB_NUMERIC_CHANGED callback, which is generated when a PtNumericInteger widget changes its value.

The Pt_CB_NUMERIC_CHANGED callback is used by both PtNumericFloatCallback and PtNumericIntegerCallback, but the type of numeric_value is double or integer, depending on where the callback was used.

Also see Pt_CB_NUMERIC_CHANGED in the PtNumericInteger section of the Photon documentation.

Instance Variables

numeric_value

An integer indicating the current value of the widget.

Associated Classes

PtNumeric, PtNumericInteger, PtNumericFloatCallback, PtCallbackInfo

PtOnOffButtonCallback

`PtOnOffButtonCallback` — state change callback information.

Synopsis

```
class PtOnOffButtonCallback
{
    state;    // integer
}
```

Description

This class holds information from the `Pt_CB_NEW_VALUE` callback, which is generated when a `PtOnOffButton` is toggled ON or OFF.

Also see Pt_CB_NEW_VALUE in the PtOnOffButton section of the Photon documentation.

Instance Variables

`state`

An integer indicating the current state of the button. 0 is OFF; 1 or any non-zero value is ON.

Associated Classes

`PtOnOffButton`, `PtCallbackInfo`

PtScrollbarCallback

PtScrollbarCallback — handle movement callback information.

Synopsis

```
class PtScrollbarCallback
{
    action;      // unsigned integer
    position;    // integer
}
```

Description

This class holds information from Pt_CB_SCROLL_MOVE (in PtScrollbar) as well as Pt_CB_SCROLLED_X and Pt_CB_SCROLLED_Y (in PtScrollArea). These callbacks are generated when the position of the handle of a scrollbar is changed.

Also see Pt_CB_SCROLL_MOVE in the PtScrollbar section, or Pt_CB_SCROLLED_X in the PtScrollArea section, of the Photon documentation.

Instance Variables

action

This instance variable monitors the action of the scrollbar handle, and takes one of the following values:

Constant	Description
Pt_SCROLL_INCREMENT	The scrollbar handle has moved down or right by one increment.
Pt_SCROLL_DECREMENT	The scrollbar handle has moved up or left by one increment.
Pt_SCROLL_PAGE_INCREMENT	The scrollbar handle has moved down or right by the equivalent of one page.
Pt_SCROLL_PAGE_DECREMENT	The scrollbar handle has moved up or left by the equivalent of one page.
Pt_SCROLL_TO_MAX	The scrollbar handle has moved to the bottom or far right.
Pt_SCROLL_TO_MIN	The scrollbar handle has moved to the top or far left.
Pt_SCROLL_DRAGGED	The scrollbar handle is being dragged.
Pt_SCROLL_RELEASED	The scrollbar handle has been released from a drag.
Pt_SCROLL_SET	The scrollbar handle position was set with a function call.

position

An integer specifying the position of the scrollbar handle.

Associated Classes

PtScrollbar, PtCallbackInfo

PtSliderCallback

`PtSliderCallback` — handle movement callback information.

Synopsis

```
class PtSliderCallback
{
    position;    // integer
}
```

Description

This class holds information from the `Pt_CB_SLIDER_MOVE` callback, which is generated when the handle of a `PtSlider` is moved.

Also see Pt_CB_SLIDER_MOVE in the PtSlider section of the Photon documentation.

Instance Variables

`position`

A number indicating the position of the new location of the slider handle.

Associated Classes

`PtSlider`, `PtCallbackInfo`

PtTerminalFontChange

PtTerminalFontChange — font change callback information.

Synopsis

```
class PtTerminalFontChange
{
    new_font;    // string
    old_font;    // string
}
```

Description

This class holds the names of old and new fonts on a generated by a Pt_CB_TERM_FONT callback.

Also see Pt_CB_TERM_FONT in the PtTerminal section of the Photon documentation.

Instance Variables

new_font

A string identifying the new font.

old_font

A string identifying the old font.

Associated Classes

PtTerminal, PtCallbackInfo

PtTerminalInput

PtTerminalInput — input callback information.

Synopsis

```
class PtTerminalInput
{
    length;      // unsigned short
    position;    // PhPoint
    string;      // string
}
```

Description

This class holds information on terminal text input generated by a Pt_CB_TERM_INPUT callback. This callback is generated for every keystroke except those where both **Alt** and **Ctrl** are held down.

Also see Pt_CB_TERM_INPUT in the PtTerminal section of the Photon documentation.

Instance Variables

length

A number of characters (usually 0 or 1) generated by the keystroke.

position

The current cursor or mouse position.

string

A string comprising the character typed.

Associated Classes

PtTerminal, PtCallbackInfo

PtTerminalOptionChange

`PtTerminalOptionChange` — option change callback information.

Synopsis

```
class PtTerminalOptionChange
{
    new_opts;    // long
    old_opts;    // long
}
```

Description

This class holds information on option changes generated by the `Pt_CB_TERM_OPTIONS` callback.

Also see Pt_CB_TERM_OPTIONS in the `PtTerminal` section of the Photon documentation.

Instance Variables

`new_opts`

A number specifying the value of the new options.

`old_opts`

A number specifying the value of the previous options.

Associated Classes

`PtTerminal`, `PtCallbackInfo`

PtTerminalScrlbkCb

`PtTerminalScrlbkCb` — scrollbar movement callback information.

Synopsis

```
class PtTerminalScrlbkCb
{
    new_count;    // short
    new_pos;      // short
    old_count;    // short
    old_pos;      // short
}
```

Description

This class holds old and new terminal scrollbar information generated by a `Pt_CB_TERM_SCRLBK` callback.

Also see Pt_CB_TERM_SCRLBK in the `PtTerminal` section of the Photon documentation.

Instance Variables

`new_count`

A number indicating the new line count.

`new_pos`

A number indicating the new line position.

`old_count`

A number indicating the former line count.

`old_pos`

A number indicating the former line position.

Associated Classes

`PtTerminal`, `PtCallbackInfo`

PtTerminalSizeChange

PtTerminalSizeChange — size change callback information.

Synopsis

```
class PtTerminalSizeChange
{
    new_size;    // PhPoint
    old_size;    // PhPoint
}
```

Description

This class holds information on size changes generated by Pt_CB_TERM_RESIZE and Pt_CB_TERM_RESIZED callbacks, as well as the Pt_CB_TTY_DEVSIZE callback used by PtTty.

For detailed information, please refer to Pt_CB_TERM_RESIZE in the PtTerminal section and Pt_CB_TTY_DEVSIZE in the PtTty section of the Photon documentation.

Instance Variables

new_size

A PhPoint indicating the new size of the terminal.

old_size

A PhPoint indicating the former size of the terminal.

Associated Classes

PtTerminal, PtTty, PtTtyOutput, PtCallbackInfo

PtTextCallback

PtTextCallback — text change callback information.

Synopsis

```
class PtTextCallback
{
    cur_insert;    // short
    doit;          // integer
    end_pos;       // short
    length;        // unsigned short
    new_insert;    // short
    start_pos;     // short
    text;          // string
}
```

Description

This class holds information from Pt_CB_MODIFY_CHANGED, Pt_CB_MODIFY_VERIFY, Pt_CB_MOTION_NOTIFY, Pt_CB_MOTION_VERIFY, and Pt_CB_TEXT_CHANGED callbacks. These callbacks are generated when text is entered in a PtText or PtComboBox widget.

Also see PtTextCallback, PtText, and PtComboBox in the Photon documentation.

Instance Variables

cur_insert

A number indicating the position of the cursor in the text string. 0 is to the left of the first character, 1 is to the left of the second character, etc.

doit

A number indicating whether a modification will be applied. 0 means no, the cursor will remain at cur_insert. Any nonzero value means yes, the cursor will move to new_insert.

end_pos

An integer indicating the position of the last character in selected text.

length

An integer indicating the length of the text string.

new_insert

A number indicating the new cursor position after any changes have been made.

start_pos

An integer indicating the position of the first character in selected text.

text

A string that corresponds to the text in the box.

Associated Classes

PtText, PtComboBox, PtCallbackInfo

PtTreeCallback

PtTreeCallback — PtTree callback information.

Synopsis

```
class PtTreeCallback
{
    click_count;    // integer
    expand;         // integer
    item;           // PtTreeItem
    nitems;         // unsigned integer
    sel_mode;       // unsigned integer
}
```

Description

This class holds information from a Pt_CB_TREE_STATE or Pt_CB_TREE_SELECTION callback, which are generated when a PtTreeItem is selected.

Also see Pt_CB_TREE_STATE and Pt_CB_TREE_SELECTION in the PtTree section of the Photon documentation.

Instance Variables

click_count

The number of times the mouse button has been clicked on the item.

expand

An integer used with Pt_CB_TREE_STATE to prevent expansion. 0 allows expansion to occur, any non-zero value prevents it.

item

A PtTreeItem that the user selected, or the first item of a range of items if the sel_mode has been set to Pt_RANGE_MODE.

nitems

The number of items selected, not including collapsed subtrees.

sel_mode

A number indicating the current selection mode. For details, see Pt_ARG_SELECTION_MODE in the PtGenList section of the Photon documentation.

Associated Classes

PtTree, PtCallbackInfo

PtTtyOutput

PtTtyOutput — output callback information.

Synopsis

```
class PtTtyOutput
{
    buffer;      // string
    length;      // unsigned short
    new_size;    // PhPoint
    old_size;    // PhPoint
}
```

Description

This class holds information from the Pt_CB_TTY_OUTPUT callback, which is generated when device output is received. The other callback for PtTty, Pt_CB_TTY_DEVSIZE, is documented with PtTerminalSizeChange.

Also see Pt_CB_TTY_OUTPUT in the PtTty section of the Photon documentation.

Instance Variables

buffer

A string identifying the buffer containing output characters.

length

The number of output characters received.

new_size

A PhPoint indicating the new size of a resized device, in rows and columns.

old_size

A PhPoint indicating the former size of a resized device, in rows and columns.

Associated Classes

PtTty, PtTerminalSizeChange, PtCallbackInfo

RtMeterCallback

`RtMeterCallback` — needle movement callback information (in QNX 4).

Synopsis

```
class RtMeterCallback
{
    position;    // integer
    severity;    // integer
}
```

Description

This class holds information from the `Rt_CB_METER_MOVED` callback, which is generated when the meter needle moves. The QNX 6 version of this widget is `PtMeterCallback`.

Also see Rt_CB_METER_MOVED in the RtMeter section of the Photon documentation.

Instance Variables

`position`

A number indicating the needle position.

`severity`

A number indicating the severity arc that the needle is in.

Associated Classes

`RtMeter`, `PtCallbackInfo`

III. Common Classes

Table of Contents

CwMatrixCell

`CwMatrixCell` — A cell of a `CwMatrix`.

Synopsis

```
class CwMatrixCell
{
    col;            // short
    colhi;          // short
    collo;          // short
    fill_color;     // color
    flags;          // flag
    font;           // string
    format;         // flag
    formula;        // string
    label;          // string
    row;            // short
    text_color;     // color
    text_string;    // string
}
```

Description

This class is not yet documented in detail.

Instance Variables

`col`

The column number of this cell.

`colhi`

Read only. The first column in which the text for this cell is drawn.

`collo`

Read only. The last column in which the text for this cell is drawn.

`fill_color`

A number specifying the fill color for this cell. Default is `0xffffff` (white).

`flags`

This instance variable controls the appearance of the cell, and may be a combination of zero or more of the following flags:

Constant	Description
`Cw_CELL_LEFT`	Aligns cell contents to the left.
`Cw_CELL_RIGHT`	Aligns cell contents to the right. This is the default.
`Cw_CELL_CENTER`	Aligns cell contents to the center.
`Cw_CELL_DECIMAL`	Aligns cell contents to the decimal point.
`Cw_CELL_JUSTIFY_MASK`	Aligns cell contents according to the justify mask.
`Cw_CELL_BORDER_LEFT_THIN`	Makes the left border thin.
`Cw_CELL_BORDER_RIGHT_THIN`	Makes the right border thin.
`Cw_CELL_BORDER_TOP_THIN`	Makes the top border thin.
`Cw_CELL_BORDER_BOTTOM_THIN`	Makes the bottom border thin.

Constant	Description
`Cw_CELL_ALL_BORDERS_THIN`	Makes all borders thin. This is the default.
`Cw_CELL_BORDER_LEFT_THICK`	Makes the left border thick.
`Cw_CELL_BORDER_RIGHT_THICK`	Makes the right border thick.
`Cw_CELL_BORDER_TOP_THICK`	Makes the top border thick.
`Cw_CELL_BORDER_BOTTOM_THICK`	Makes the bottom border thick.
`Cw_CELL_ALL_BORDERS_THICK`	Makes all borders thick.
`Cw_CELL_BORDER_LEFT`	Puts a border on the left side of the cell.
`Cw_CELL_BORDER_RIGHT`	Puts a border on the right side of the cell.
`Cw_CELL_BORDER_TOP`	Puts a border on the top of the cell.
`Cw_CELL_BORDER_BOTTOM`	Puts a border on the bottom of the cell.
`Cw_CELL_ALL_BORDERS`	Puts a border on both sides, top, and bottm of the cell. This is the default.
`Cw_CELL_INVERTED`	Inverts the colors of the cell, exchanging the `fill_color` for the `text_color`, and vise versa.
`Cw_CELL_SELECTION_BORDER`	Read only.
`Cw_CELL_COVERED`	Read only.

`font`

A string specifying the font used in the cell. Default is `helv12`.

`format`

A number, ignored by the widget, but available as a place for the programmer to store information.

`formula`

A string, ignored by the widget, but available as a place for the programmer to store information.

`label`

A string, ignored by the widget, but available as a place for the programmer to store information.

`row`

The row number of this cell.

`text_color`

A number specifying the text color for this cell. Default is `0x0` (black).

`text_string`

A string comprising the displayed text of the cell.

Associated Classes

`CwMatrix`, `CwMatrixCallback`

PhArea

PhArea — The position and size of a rectangular area.

Synopsis

```
class PhArea
{
    pos;     // PhPoint
    size;    // PhDim
}
```

Description

This class defines an area by its position (*pos*) and *size*. The position is relative to the upper left-hand corner of the containing widget. The units of all measurements are pixels.

The only reliable way to set or change the dim, pos, and area variables of widgets (not other classes) is with SetDim, SetPos and SetArea variables of PtWidget. Attempting to change these variables or their sub-components without using these methods will lead to unpredictable results in your program.

Do not confuse the size variable of a PhArea with the dim variable of a PtWidget. They are both instances of PhDim, but the former belongs to a common class, while the latter belongs to a widget class.

Also see PhArea in the Photon documentation.

Instance Variables

pos

A PhPoint specifying the upper-left hand corner position of the area.

size

A PhDim specifying the height and width of the area.

Example

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

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

/*
This example puts up a window with a rectangle in it, and
prints the rectangle's area, position, and dimensions.
*/

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

win = new(PtWindow);
win.SetDim(200,100);
win.SetPos(350,30);

rect = new(PtRect);
rect.SetArea(18,22,150,50);
rect.fill_color = PgRGB(100,220,240);

pretty_princ("The area is: \n",rect.area,"\n\n");
princ("The position is: \n",rect.pos,"\n\n");
princ("The dimensions are: \n",rect.dim,"\n");
```

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

PhBlitEvent

`PhBlitEvent` — A bit block transformation (blit) event on a rectangular area.

Synopsis

```
class PhBlitEvent
{
    offset;    // PhPoint
    rect;      // PhRect
    rid;       // region ID
}
```

Description

Working with this class is beyond the scope of normal Gamma and Photon programming. For more information, please refer to PhEvent in the Photon documentation.

PhBoundaryEvent

PhBoundaryEvent — The event of leaving or entering an event region.

Synopsis

```
class PhBoundaryEvent
{
    entered;    // PhEventRegion
    left;       // PhEventRegion
}
```

Description

Working with this class is beyond the scope of normal Gamma and Photon programming. For more information, please refer to PhEvent in the Photon documentation.

PhDim

PhDim — The dimensions (height and width) of a widget.

Synopsis

```
class PhDim
{
    h;    // short
    w;    // short
}
```

Description

This class defines the dimensions of an area by height (*h*) and width (*w*). The units of measure are pixels.

The only reliable way to set or change the dim, pos, and area variables of widgets (not other classes) is with SetDim, SetPos and SetArea variables of PtWidget. Attempting to change these variables or their sub-components without using these methods will lead to unpredictable results in your program.

Also see PhDim in the Photon documentation.

Instance Variables

h

An integer specifying the height of an area in pixels.

w

An integer specifying the width of an area in pixels.

Example

See the example for PhArea.

PhDragEvent

`PhDragEvent` — The event of dragging an object.

Synopsis

```
class PhDragEvent
{
    boundary;    // PhRect
    flags;       // unsigned short
    max;         // PhDim
    min;         // PhDim
    rect;        // PhRect
    rid;         // region ID
    step;        // PhDim
}
```

Description

Working with this class is beyond the scope of normal Gamma and Photon programming. For more information, please refer to PhEvent in the Photon documentation.

PhDrawEvent

PhDrawEvent — The event of drawing an object.

Synopsis

```
class PhDrawEvent
{
    cmd_buffer_size;       // unsigned short
    draw_event_version;    // unsigned short
    id;                    // unsigned long
}
```

Description

Working with this class is beyond the scope of normal Gamma and Photon programming. For more information, please refer to PhEvent in the Photon documentation.

PhEvent

`PhEvent` — An event.

Synopsis

```
class PhEvent
{
    collector;          // PhEventRegion
    data_len;           // unsigned short
    emitter;            // PhEventRegion
    flags;              // unsigned short
    input_group;        // unsigned short
    num_rects;          // unsigned short
    processing_flags;   // unsigned short
    subtype;            // unsigned short
    timestamp;          // unsigned long
    translation;        // PhPoint
    type;               // unsigned long
}
```

Description

Working with this class is beyond the scope of normal Gamma and Photon programming. For more information, please refer to PhEvent in the Photon documentation.

PhEventRegion

`PhEventRegion` — A region that emits or collects an event.

Synopsis

```
class PhEventRegion
{
    handle;    // long
    rid;       // unsigned short
}
```

Description

Working with this class is beyond the scope of normal Gamma and Photon programming. For more information, please refer to PhEvent in the Photon documentation.

PhImage

PhImage — Information about an image.

Synopsis

```
class PhImage
{
    bpl;            // integer
    colors;         // integer
    flags;          // char
    format;         // char
    image_tag;      // long
    palette;        // array
    palette_tag;    // long
    size;           // PhDim
    type;           // integer
    xscale;         // integer
    yscale;         // integer
}
```

Description

This class holds information about an image. Any image used in Photon must be an instance of this class.

For detailed information, please refer to PhImage in the Photon documentation.

Instance Variables

`bpl`

An integer specifying the number of bytes per line of the image.

`colors`

An integer specifying the number of colors of the image.

`flags`

This instance variable may be a combination of zero or more of the following flags:

Constant	Description
Pt_HIGHLIGHTED	Draw a beveled border around the widget.
Pt_AUTOHIGHLIGHT	Automatically display/remove the highlight border as the cursor passes over the widget.
Pt_ETCH_HIGHLIGHT	Draw a double bevel if Pt_HIGHLIGHTED is on.
Pt_SET	Make the widget 'set'. This will invert the coloring on the bevel as well.
Pt_TOGGLE	The widget's SET flag will toggle on each mouse click instead of changing with mouse down/mouse up events.
Pt_SELECTABLE	The widget may be selected, causing Pt_CB_ARM/ACTIVATE/DISARM events.
Pt_GHOST	The widget is displayed 'ghosted'. This does not affect its response to events.
Pt_BLOCKED	The widget will be unresponsive to events.
Pt_REALIZED	The widget is visible on the display.

Constant	Description
Pt_CLIP_HIGHLIGHT	The corners of the highlight rectangle are clipped off.
Pt_OPAQUE	Makes the widget opaque within its own extent and for everything behind it.
Pt_DELAY_REALIZE	The widget won't be realized except by a call to PtRealizeWidget().
Pt_GETS_FOCUS	The widget may get the keyboard focus.
Pt_MENU_BUTTON	Makes the widget a menu item.
Pt_DESTROYED	Marks the widget for destruction.
Pt_DAMAGED	Marks the widget for repairs.
Pt_OBSCURED	The widget is covered by another widget, or is outside its parent's canvas.
Pt_IN_FLUX	A call to PtContainerHold has been made on the widget.
Pt_CLEAR	Keeps the widget's extent clear of any brothers in front of it.
Pt_DAMAGE_FAMILY	The widget and its children will be repaired.
Pt_SELECT_NOREDRAW	The widget will not redraw when it is selected.
Pt_WIDGET_REBUILD	The widget will be rebuilt once all resources have been changed.
Pt_WIDGET_RESIZE	The widget will be resized once all resources have been changed.
Pt_PROCREATED	The widget is a procreated child of a compound widget.
Pt_ALL_BUTTONS	The widget treats events on any mouse button as a selection.
Pt_FOCUS_RENDER	The widget will attempt to indicate that it has the keyboard focus through some means.
Pt_CALLBACKS_ACTIVE	Callbacks for this widget will be called due to changes through code, not just due to user interactions.
Pt_MENUABLE	This widget will respond to the menu button with a Pt_CB_MENU event.
Pt_NOREDRAW_SET	NoRedraw Set
Pt_FREE_MEMORY	Frees memory associated with widget pointers.
Pt_REGION	Force the widget to have a region ID.
Pt_REALIZING	The widget is being realized.

`format`

Not in use.

`image_tag`

A data tag used for image caching. A value is automatically assigned to this variable when you create an image in PhAB, or load an image with `PxLoadImage`.

`palette`

The image palette.

`palette_tag`

A tag for the palette data.

`size`

A `PhDim` specifying the size of the image.

`type`

The type of graphic. See `PgDrawImage` in the Photon documentation for details.

`xscale`

An integer specifying the horizontal scaling factor.

`yscale`

An integer specifying the vertical scaling factor.

PhKeyEvent

`PhKeyEvent` — The event of pressing a keyboard key.

Synopsis

```
class PhKeyEvent
{
    button_state;    // unsigned short
    key_cap;         // unsigned long
    key_flags;       // unsigned long
    key_mods;        // unsigned long
    key_scan;        // unsigned char
    key_sym;         // unsigned long
    pos;             // PhPoint
}
```

Description

Working with this class is beyond the scope of normal Gamma and Photon programming. For more information, please refer to PhEvent in the Photon documentation.

PhPoint, PhLPoint

PhPoint, PhPoint — The (x,y) coordinates of a point.

Synopsis

```
class PhPoint
{
    x;    // short
    y;    // short
}

class PhLPoint
{
    x;    // long
    y;    // long
}
```

Description

These classes define a point by its *x* and *y* coordinates. For PhPoint the coordinates are expressed as integers, while for PhLPoint they are expressed as longs. The units of measure are pixels.

Also see PhPoint in the Photon documentation.

Instance Variables

x

An integer or long specifying the number of pixels to the right (horizontally) from the upper-left corner of the containing widget.

y

An integer or long specifying the number of pixels down (vertically) from the upper-left corner of the containing widget.

Example

See the example for PtPixel.

PhPointerEvent

`PhPointerEvent` — An event initiated by the pointer.

Synopsis

```
class PhPointerEvent
{
    button_state;    // unsigned short
    buttons;         // unsigned short
    click_count;     // unsigned short
    flags;           // unsigned char
    key_mods;        // unsigned short
    pos;             // PhPoint
    z;               // short
}
```

Description

Working with this class is beyond the scope of normal Gamma and Photon programming. For more information, please refer to PhEvent in the Photon documentation.

PhRect

`PhRect` — A rectangular area defined by two points.

Synopsis

```
class PhRect
{
    lr;    // PhPoint
    ul;    // PhPoint
}
```

Description

This class defines a rectangular area by the upper left (*ul*) and lower right (*lr*) corner points. The position of both points is relative to the upper left-hand corner of the containing widget. The units of all measurements are pixels.

Also see PhRect in the Photon documentation.

Instance Variables

`lr`

A `PhPoint` specifying the lower right corner of the rectangular area.

`ul`

A `PhPoint` specifying the upper left corner of the rectangular area.

PhRegion

`PhRegion` — An area of the interface.

Synopsis

```
class PhRegion
{
    bro_behind;       // region ID
    bro_in_front;     // region ID
    child;            // region ID
    cursor_color;     // color
    cursor_type;      // unsigned short
    data_len;         // unsigned short
    events_opaque;    // unsigned long
    events_sense;     // unsigned long
    flags;            // unsigned long
    handle;           // long
    input_group;      // unsigned short
    origin;           // PhPoint
    owner;            // short
    parent;           // region ID
    rid;              // region ID
    state;            // unsigned short
}
```

Description

This class describes a region. Regions are abstract areas of the interface that may be sensitve to various events, such as keyboard input, mouse actions, or other window events.

For detailed information, please refer to PhRegion in the Photon documentation.

Instance Variables

`bro_behind`

A region ID specifying the brother region immediately behind this region. A value of -1 indicates no brother region is behind.

`bro_in_front`

A region ID specifying the brother region immediately in front of this region. A value of -1 indicates no brother region is in front.

`child`

A region ID specifying the foremost child region of this region. A value of -1 indicates there is no child region.

`cursor_color`

A number specifying the color for the cursor when in this region. Default is `0` (white).

`cursor_type`

This instance variable specifies the appearance of the cursor when in this region, and may have one of the following values:

Constant	Description
`Ph_CURSOR_NO_INHERIT`	Do not inherit the parent's cursor.
`Ph_CURSOR_MANUAL_CONTROL`	Not implemented.
`Ph_CURSOR_NONE`	Do not show a cursor.

Constant	Description
Ph_CURSOR_BITMAP	Not implemented.
Ph_CURSOR_INHERIT	Inherit the parent's cursor.
Ph_CURSOR_POINTER	Pointer
Ph_CURSOR_BIG_POINTER	Big Pointer
Ph_CURSOR_MOVE	Move
Ph_CURSOR_CROSSHAIR	Crosshair
Ph_CURSOR_CLOCK	Clock
Ph_CURSOR_WAIT	Wait
Ph_CURSOR_NOINPUT	No Input
Ph_CURSOR_DONT	Don't
Ph_CURSOR_FINGER	Finger
Ph_CURSOR_INSERT	Insert
Ph_CURSOR_DRAG_VERTICAL	Drag Vertical
Ph_CURSOR_DRAG_TOP	Drag Top
Ph_CURSOR_DRAG_BOTTOM	Drag Bottom
Ph_CURSOR_DRAG_HORIZONTAL	Drag Horizontal
Ph_CURSOR_DRAG_LEFT	Drag Left
Ph_CURSOR_DRAG_RIGHT	Drag Right
Ph_CURSOR_DRAG_BACKDIAG	Drag Backdiag
Ph_CURSOR_DRAG_TL	Drag Top Left
Ph_CURSOR_DRAG_BR	Drag Bottom Right
Ph_CURSOR_DRAG_FOREDIAG	Drag Foreward Diagonal
Ph_CURSOR_DRAG_TR	Drag Top Right
Ph_CURSOR_DRAG_BL	Drag Bottom Left
Ph_CURSOR_POINT_WAIT	Point Wait
Ph_CURSOR_LONG_WAIT	Long Wait
Ph_CURSOR_QUESTION_POINT	Question Point
Ph_CURSOR_PASTE	Paste

`data_len`

An integer specifying the length of data for this region.

`events_opaque`

A number specifying the events that this region is opaque to. The region will clip out and ignore any part of such an event as it passes through.

`events_sense`

A number specifying the events that this region is sensitive to. The region will respond to and pass on to the application any such event that passes through.

`flags`

This instance variable may be a combination of zero or more of the following flags:

Constant	Description
Pt_HIGHLIGHTED	Draw a beveled border around the widget.

Constant	Description
Pt_AUTOHIGHLIGHT	Automatically display/remove the highlight border as the cursor passes over the widget.
Pt_ETCH_HIGHLIGHT	Draw a double bevel if Pt_HIGHLIGHTED is on.
Pt_SET	Make the widget 'set'. This will invert the coloring on the bevel as well.
Pt_TOGGLE	The widget's SET flag will toggle on each mouse click instead of changing with mouse down/mouse up events.
Pt_SELECTABLE	The widget may be selected, causing Pt_CB_ARM/ACTIVATE/DISARM events.
Pt_GHOST	The widget is displayed 'ghosted'. This does not affect its response to events.
Pt_BLOCKED	The widget will be unresponsive to events.
Pt_REALIZED	The widget is visible on the display.
Pt_CLIP_HIGHLIGHT	The corners of the highlight rectangle are clipped off.
Pt_OPAQUE	Makes the widget opaque within its own extent and for everything behind it.
Pt_DELAY_REALIZE	The widget won't be realized except by a call to PtRealizeWidget().
Pt_GETS_FOCUS	The widget may get the keyboard focus.
Pt_MENU_BUTTON	Makes the widget a menu item.
Pt_DESTROYED	Marks the widget for destruction.
Pt_DAMAGED	Marks the widget for repairs.
Pt_OBSCURED	The widget is covered by another widget, or is outside its parent's canvas.
Pt_IN_FLUX	A call to PtContainerHold has been made on the widget.
Pt_CLEAR	Keeps the widget's extent clear of any brothers in front of it.
Pt_DAMAGE_FAMILY	The widget and its children will be repaired.
Pt_SELECT_NOREDRAW	The widget will not redraw when it is selected.
Pt_WIDGET_REBUILD	The widget will be rebuilt once all resources have been changed.
Pt_WIDGET_RESIZE	The widget will be resized once all resources have been changed.
Pt_PROCREATED	The widget is a procreated child of a compound widget.
Pt_ALL_BUTTONS	The widget treats events on any mouse button as a selection.
Pt_FOCUS_RENDER	The widget will attempt to indicate that it has the keyboard focus through some means.
Pt_CALLBACKS_ACTIVE	Callbacks for this widget will be called due to changes through code, not just due to user interactions.

Constant	Description
Pt_MENUABLE	This widget will respond to the menu button with a Pt_CB_MENU event.
Pt_NOREDRAW_SET	Noredraw Set
Pt_FREE_MEMORY	Frees memory associated with widget pointers.
Pt_REGION	Force the widget to have a region ID.
Pt_REALIZING	The widget is being realized.

handle

A user-defined number for passing small amounts of information along with events.

input_group

A number specifying the input group. If this region is not an input group, 0.

origin

A PhPoint specifying the region's origin, with respect to the origin of it's parent. This is the reference point for all other coordinates in this region.

owner

A number indicating the owner of this region.

parent

The region ID of the parent of this region.

rid

The ID number of this region, which is assigned when the region is opened.

state

This instance variable controls the state of the region, and may have one of the following values:

Constant	Description
Ph_WM_STATE_ISMASK	Internal informational bit.
Ph_WM_STATE_ISHIDDEN	The window opens normally, but is not displayed.
Ph_WM_STATE_ISMAX	Opens the window to its maximum size.
Ph_WM_STATE_ISBACKDROP	Opens the window as a workspace backdrop.
Ph_WM_STATE_ISTASKBAR	Internal informational bit.
Ph_WM_STATE_ISPDM	Internal informational bit.
Ph_WM_STATE_ISICONIFIED	Opens the window and makes it an icon.
Ph_WM_STATE_ISICON	Internal informational bit.
Ph_WM_STATE_ISFRONT	Opens the window in front of all other applications.
Ph_WM_STATE_ISFOCUS	Opens the window as the focus widget, if the WM cursor focus option is disabled.

PhWindowEvent

`PhWindowEvent` — An event related to a window.

Synopsis

```
class PhWindowEvent
{
    event_f;        // unsigned short
    event_state;    // short
    pos;            // PhPoint
    rid;            // region ID
    size;           // PhPoint
    state_f;        // unsigned short
}
```

Description

Working with this class is beyond the scope of normal Gamma and Photon programming. For more information, please refer to PhEvent in the Photon documentation.

PtCalendarDate

`PtCalendarDate` — the day, month, and year.

Synopsis

```
class PtCalendarDate
{
    day;      // char
    month;    // char
    year;     // short
}
```

Description

This class holds calendar date information: the day, month and year, used in the `PtCalendar` widget. The values for `day` and `month` start with 0, so they are always 1 less than the actual date or month number. For example, April 21, 2000 is expressed as:

```
{PtCalendarDate (day . 20) (month . 3) (year . 2000)}
```

Instance Variables

`day`

An integer ranging from 0 - 30 specifying the day of the month.

`month`

An integer ranging from 0 - 11 specifying the month of year.

`year`

An integer ranging from -32767 - +32767 specifying the year.

Associated Classes

`PtCalendar`, `PtCalendarSelectCallback`

PtEventData

`PtEventData` — data related to a variety of events.

Synopsis

```
class PtEventData
{
    blit_event;         // PhBlitEvent
    boundary_event;     // PhBoundaryEvent
    drag_event;         // PhDragEvent
    draw_event;         // PhDrawEvent
    key_event;          // PhKeyEvent
    pointer_event;      // PhPointerEvent
    rect;               // PhRect
    window_event;       // PhWindowEvent
}
```

Description

Working with this class is beyond the scope of normal Gamma and Photon programming. For more information, please refer to PhEvent in the Photon documentation.

PtFileSelItem

PtFileSelItem — an item in a PtFileSel.

Synopsis

```
class PtFileSelItem
{
    fullpath;     // string
    opened;       // short
    otherinfo;    // string
    root;         // short
    type;         // short
}
```

Description

This class is an item in a PtFileSel.

Also see PtFileSel in the Photon documentation.

Instance Variables

fullpath

A string specifying the item's pathname.

opened

This variable indicates whether or not the children of the directory have been allocated, and can have one of the following values.

Constant	Description
Pt_FS_NEW_DIR	An unallocated directory.
Pt_FS_OLD_DIR	An allocated directory.

otherinfo

A string that holds other information.

root

A number indicating whether this item is the root directory. 1 means yes, 0 means no.

type

This variable indicates the type of item, and can have one of the following values.

Constant	Description
Pt_FS_DIR_OP	An open directory (items visible).
Pt_FS_DIR_CL	A closed directory (items not visible).
Pt_FS_DLINK_OP	A link to an open directory.
Pt_FS_DLINK_CL	A link to a closed directory.
Pt_FS_FILE	A file.
Pt_FS_FLINK	A link to a file.
Pt_FS_ERROR	A file that had a read error.

Associated Classes

PtFileSel

PtMultiTextAttributes

PtMultiTextAttributes — the attributes of PtMultiText text.

Synopsis

```
class PtMultiTextAttributes
{
    background_color;    // color
    flags;               // integer
    font;                // string
    text_color;          // color
}
```

Description

This class holds information that PtMultiText uses for formatting and displaying text.

Also see PtMultiTextAttributes in the Photon documentation.

Instance Variables

background_color

Either a number specifying a background color, *or* one of the following values:

Pt_INHERIT_COLOR	Specifies the same background color as the previous range.
Pt_DEFAULT_COLOR	Specifies the value of Pt_ARG_COLOR.

flags

This instance variable controls text characteristics, and may be a combination of zero or more of the following flags:

Constant	Description
Pt_HIGHLIGHTED	Draw a beveled border around the widget.
Pt_AUTOHIGHLIGHT	Automatically display/remove the highlight border as the cursor passes over the widget.
Pt_ETCH_HIGHLIGHT	Draw a double bevel if Pt_HIGHLIGHTED is on.
Pt_SET	Make the widget 'set'. This will invert the coloring on the bevel as well.
Pt_TOGGLE	The widget's SET flag will toggle on each mouse click instead of changing with mouse down/mouse up events.
Pt_SELECTABLE	The widget may be selected, causing Pt_CB_ARM/ACTIVATE/DISARM events.
Pt_GHOST	The widget is displayed 'ghosted'. This does not affect its response to events.
Pt_BLOCKED	The widget will be unresponsive to events.
Pt_REALIZED	The widget is visible on the display.

Constant	Description
Pt_CLIP_HIGHLIGHT	The corners of the highlight rectangle are clipped off.
Pt_OPAQUE	Makes the widget opaque within its own extent and for everything behind it.
Pt_DELAY_REALIZE	The widget won't be realized except by a call to PtRealizeWidget.
Pt_GETS_FOCUS	The widget may get the keyboard focus.
Pt_MENU_BUTTON	Makes the widget a menu item.
Pt_DESTROYED	Marks the widget for destruction.
Pt_DAMAGED	Marks the widget for repairs.
Pt_OBSCURED	The widget is covered by another widget, or is outside its parent's canvas.
Pt_IN_FLUX	A call to PtContainerHold has been made on the widget.
Pt_CLEAR	Keeps the widget's extent clear of any brothers in front of it.
Pt_DAMAGE_FAMILY	The widget and its children will be repaired.
Pt_SELECT_NOREDRAW	The widget will not redraw when it is selected.
Pt_WIDGET_REBUILD	The widget will be rebuilt once all resources have been changed.
Pt_WIDGET_RESIZE	The widget will be resized once all resources have been changed.
Pt_PROCREATED	The widget is a procreated child of a compound widget.
Pt_ALL_BUTTONS	The widget treats events on any mouse button as a selection.
Pt_FOCUS_RENDER	The widget will attempt to indicate that it has the keyboard focus through some means.
Pt_CALLBACKS_ACTIVE	Callbacks for this widget will be called due to changes through code, not just due to user interactions.
Pt_MENUABLE	This widget will respond to the menu button with a Pt_CB_MENU event.
Pt_NOREDRAW_SET	Noredraw Set
Pt_FREE_MEMORY	Frees memory associated with widget pointers.
Pt_REGION	Force the widget to have a region ID.
Pt_REALIZING	The widget is being realized.

font

Either a string specifying the name of a font, *or* one of the following values:

Pt_INHERIT_FONT	Specifies the same font as the previous range.
Pt_DEFAULT_FONT	Specifies the value of Pt_ARG_TEXT_FONT.

`text_color`

Either a number specifying a font color, *or* one of the following values:

`Pt_INHERIT_COLOR`	Specifies the same font color as the previous range.
`Pt_DEFAULT_COLOR`	Specifies the value of `Pt_ARG_COLOR`.

PtTreeItem

PtTreeItem — an item in a PtTree.

Synopsis

```
class PtTreeItem
{
    flags;        // unsigned long
    set_img;      // short
    string;       // string
    unset_img;    // short
}
```

Description

This class holds information about an item in a PtTree.

The images for set_img and unset_img are located in the Pt_ARG_TREE_IMAGES resource, which is described in detail in the Photon documentation.

Also see PtTreeItem in the Photon documentation.

Instance Variables

flags

This instance variable controls the widget's characteristics, and may be a combination of zero or more of the following flags:

Constant	Description
Pt_HIGHLIGHTED	Draw a beveled border around the widget.
Pt_AUTOHIGHLIGHT	Automatically display/remove the highlight border as the cursor passes over the widget.
Pt_ETCH_HIGHLIGHT	Draw a double bevel if Pt_HIGHLIGHTED is on.
Pt_SET	Make the widget 'set'. This will invert the coloring on the bevel as well.
Pt_TOGGLE	The widget's SET flag will toggle on each mouse click instead of changing with mouse down/mouse up events.
Pt_SELECTABLE	The widget may be selected, causing Pt_CB_ARM/ACTIVATE/DISARM events.
Pt_GHOST	The widget is displayed 'ghosted'. This does not affect its response to events.
Pt_BLOCKED	The widget will be unresponsive to events.
Pt_REALIZED	The widget is visible on the display.
Pt_CLIP_HIGHLIGHT	The corners of the highlight rectangle are clipped off.
Pt_OPAQUE	Makes the widget opaque within its own extent and for everything behind it.
Pt_DELAY_REALIZE	The widget won't be realized except by a call to PtRealizeWidget().
Pt_GETS_FOCUS	The widget may get the keyboard focus.

Constant	Description
Pt_MENU_BUTTON	Makes the widget a menu item.
Pt_DESTROYED	Marks the widget for destruction.
Pt_DAMAGED	Marks the widget for repairs.
Pt_OBSCURED	The widget is covered by another widget, or is outside its parent's canvas.
Pt_IN_FLUX	A call to PtContainerHold() has been made on the widget.
Pt_CLEAR	Keeps the widget's extent clear of any brothers in front of it.
Pt_DAMAGE_FAMILY	The widget and its children will be repaired.
Pt_SELECT_NOREDRAW	The widget will not redraw when it is selected.
Pt_WIDGET_REBUILD	The widget will be rebuilt once all resources have been changed.
Pt_WIDGET_RESIZE	The widget will be resized once all resources have been changed.
Pt_PROCREATED	The widget is a procreated child of a compound widget.
Pt_ALL_BUTTONS	The widget treats events on any mouse button as a selection.
Pt_FOCUS_RENDER	The widget will attempt to indicate that it has the keyboard focus through some means.
Pt_CALLBACKS_ACTIVE	Callbacks for this widget will be called due to changes through code, not just due to user interactions.
Pt_MENUABLE	This widget will respond to the menu button with a Pt_CB_MENU event.
Pt_NOREDRAW_SET	Noredraw Set
Pt_FREE_MEMORY	Frees memory associated with widget pointers.
Pt_REGION	Force the widget to have a region ID.
Pt_REALIZING	The widget is being realized.

`set_img`

The index number of an image to be displayed when the item is set.

`string`

A string comprising the text of the item.

`unset_img`

The index number of an image to be displayed when the item is not set.

Index

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

R

S

T

U

V

W

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.