Contents

About This Manual xiii
  What you’ll find in this guide xv
  Typographical conventions xvi
    Note to Windows users xvii
  Technical support xvii
What’s new in Photon for QNX Neutrino 6.3.0 Service Pack 1 xvii
What’s new in Photon for QNX Neutrino 6.3 xvii
    Resource editor plugin API xvii
What’s new in Photon for QNX Neutrino 6.2.1 xviii
    Errata xviii
What’s new in Photon for QNX Neutrino 6.2.0 xviii

1 Overview 1
  Subclassing widgets 3
  Design considerations 4
  A quick look at a widget 5
    Widget header file 5
    Widget source file 6
  Types of widgets 8
    Class hierarchy 9
    Basic widgets 9
    Container widgets 9
    Compound widgets 10

2 Life Cycle of a Widget 11
  All widgets 13
  Basic widgets 13
    Instantiating the widget 14
    Realizing a widget instance 16
    After realizing a widget 20
    Destroying a widget 21
  Container widgets 21
    Child constraints 22
3 Anatomy of a Widget 25

Defining resources 27
Resource manifests 27
PtResourceRec_t resource records 28

Defining the widget class 36
Widget class structure 36
Widget class resource table 39
PtBasic class resource additions 41

Class methods 42
Defaults method 43
Initialization method 44
Extent method 45
Connection method 46
Realization method 48
Draw method 48
Unrealization method 50
Destruction method 51
Set Resources method 51
Get Resources method 51
Got Focus method 52
Lost Focus method 52
Calc Opaque Rect method 53

Widget actions 54
Raw callback list 55

Container widget anatomy 56
Child-constraint support 56
Fundamental methods 60

Compound widget anatomy 62
Blocking resources 66
Compound redirection 66
Fundamental methods 67

4 Using Widget Superclasses 71
PtWidget 73
Class hierarchy 73
Class flags 73
Methods 73
5 Creating a List Widget 107
Overview 109
Item structure 109
Widget instance structure 110
Methods 112
List Draw method 112
List Mouse method 112
List Key method 113
List Select method 114
List Inflate method 114
List Attributes method 115
Convenience functions 115
PtSuperClassGenListDraw() 118
PtSuperClassGenListInflate() 119
PtSuperClassGenListKey() 121
PtSuperClassGenListMouse() 123
PtSuperClassGenListSelect() 125

6 Creating a Tree Widget 127
Overview 129
Item structure 129
Methods 130
Tree Draw Item method 130
Tree Item State method 131
Convenience functions 131
7  Binding Widgets into PhAB  137

Widget design considerations  139
   Single-value resources  139
   When to add custom widgets to palette files  139
   Displaying your custom widget  140
Creating a shared object  140
Creating a template  140
Editing palette.def  140
Creating a widget description table  141
   Widget class name (w=)  142
   Widget hierarchy (h=)  142
   List of resources (r=)  143
   List of callbacks (c=, k=, e=)  146
   Inherited resources and callbacks (i=, n=)  147
   Base widget (b=)  148
   Change class definition (q=)  148
   Create-type definition (t=)  148
   Cursor styles (s=)  149
   Default values (d=)  149

8  Widget Building Library API  151

PhWindowChange()  157
PhWindowClose()  159
PhWindowOpen()  160
PtAddWidgetData()  162
PtAnchorDeregister()  163
PtAnchorRegister()  164
PtAnchorWidget()  165
PtApplyAnchors()  166
PtAttemptResize()  167
PtCalcAnchorOffsets()  169
PtCalcRegion()  170
PtChildBoundingBox()  171
PtClipAdd()  172
PtClipRemove()  173
PtCompoundRedirect()  174
PtCoreChangeRegion()  175
PtCreateWidgetClass()  176
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>PtDamageExposed()</td>
<td>178</td>
</tr>
<tr>
<td>PtDestroyCallbackList()</td>
<td>179</td>
</tr>
<tr>
<td>PtDestroyHotkeyCallbacks()</td>
<td>180</td>
</tr>
<tr>
<td>PtDestroyRawCallbacks()</td>
<td>181</td>
</tr>
<tr>
<td>PtFindNextWidgetData()</td>
<td>182</td>
</tr>
<tr>
<td>PtFindResource()</td>
<td>183</td>
</tr>
<tr>
<td>PtFindWidgetData()</td>
<td>185</td>
</tr>
<tr>
<td>PtGetCallbackList()</td>
<td>186</td>
</tr>
<tr>
<td>PtGetStruct()</td>
<td>187</td>
</tr>
<tr>
<td>PtInvokeCallbackList()</td>
<td>189</td>
</tr>
<tr>
<td>PtInvokeCallbackType()</td>
<td>190</td>
</tr>
<tr>
<td>PtInvokeResizeCallbacks()</td>
<td>191</td>
</tr>
<tr>
<td>PtMoveResizeWidget()</td>
<td>192</td>
</tr>
<tr>
<td>PtRemoveWidgetData()</td>
<td>193</td>
</tr>
<tr>
<td>PtResizeCanvas()</td>
<td>194</td>
</tr>
<tr>
<td>PtResizePolicy()</td>
<td>195</td>
</tr>
<tr>
<td>PtSetExtentFromArea()</td>
<td>196</td>
</tr>
<tr>
<td>PtSetStruct()</td>
<td>197</td>
</tr>
<tr>
<td>PtSetValue()</td>
<td>199</td>
</tr>
<tr>
<td>PtSuperClassCalcOpaque()</td>
<td>200</td>
</tr>
<tr>
<td>PtSuperClassChildCreated()</td>
<td>201</td>
</tr>
<tr>
<td>PtSuperClassChildDestroyed()</td>
<td>202</td>
</tr>
<tr>
<td>PtSuperClassChildGettingFocus()</td>
<td>203</td>
</tr>
<tr>
<td>PtSuperClassChildGettingResources()</td>
<td>204</td>
</tr>
<tr>
<td>PtSuperClassChildLosingFocus()</td>
<td>205</td>
</tr>
<tr>
<td>PtSuperClassChildMovedResized()</td>
<td>206</td>
</tr>
<tr>
<td>PtSuperClassChildRealized()</td>
<td>208</td>
</tr>
<tr>
<td>PtSuperClassChildSettingResources()</td>
<td>209</td>
</tr>
<tr>
<td>PtSuperClassChildUnrealized()</td>
<td>210</td>
</tr>
<tr>
<td>PtSuperClassConnect(), PtSuperClassConnectFrom()</td>
<td>211</td>
</tr>
<tr>
<td>PtSuperClassDraw()</td>
<td>212</td>
</tr>
<tr>
<td>PtSuperClassExtent()</td>
<td>214</td>
</tr>
<tr>
<td>PtSuperClassGetResources()</td>
<td>216</td>
</tr>
<tr>
<td>PtSuperClassGotFocus()</td>
<td>217</td>
</tr>
<tr>
<td>PtSuperClassInit(), PtSuperClassInitFrom()</td>
<td>218</td>
</tr>
<tr>
<td>PtSuperClassLostFocus()</td>
<td>219</td>
</tr>
<tr>
<td>PtSuperClassRawEvent(), PtSuperClassRawEventFrom()</td>
<td>220</td>
</tr>
<tr>
<td>PtSuperClassRealized()</td>
<td>221</td>
</tr>
<tr>
<td>PtSuperClassSetResources()</td>
<td>222</td>
</tr>
<tr>
<td>PtUpdateVisibility()</td>
<td>224</td>
</tr>
<tr>
<td>PtWidgetAbove()</td>
<td>226</td>
</tr>
</tbody>
</table>
9 Creating Custom Resource Editors 227

Resource datatypes 230
The life cycle of a plugin instance 239
The res_editors.def file 242
Endian issues 244

10 The resource editor API 247

Resource Editor API structures 249
Functions exported from PhAB 250

PhABResExportCommon_t.alloc_image() 252
PhABResExportCommon_t.alloc_mem() 253
PhABResExportFull_t.answer_changes() 254
PhABResExportCommon_t.apply() 255
PhABResExportFull_t.closing() 257
PhABResExportCommon_t.cmp() 258
PhABResExportFull_t.create_window() 259
PhABResExportFull_t.destroy() 261
PhABResExportCommon_t.dup() 262
PhABResExportCommon_t.free() 263
PhABResExportCommon_t.free_image() 264
PhABResExportCommon_t.free_mem() 265
PhABResExportFull_t.get_area() 266
PhABResExportCommon_t.get_res() 267
PhABResExportCommon_t.message() 269
PhABResExportFrugal_t.open() 270
PhABResExportCommon_t 271
PhABResExportFrugal_t 273
PhABResExportFull_t 274
ResPlugin_t 275
ResPluginAnyChangesF_t 276
ResPluginBlockF_t 278
ResPluginCommon_t 279
ResPluginDestroyF_t 281
ResPluginDisableF_t 282
ResPluginFrugalCreateF_t 283
ResPluginFrugalEditor_t 285
ResPluginFullCreateF_t 286
ResPluginFullEditor_t 289
ResPluginGetAreaF_t 291
ResPluginGetChangesF_t 292
ResPluginLoadingF_t 293
ResPluginSetDataF_t 294
ResPluginToFrontF_t 296
ResPluginUnloadingF_t 297
PhABResExportFull_t.set_state() 298
PhABResExportFull_t.to_front() 300

11 Resource editor plugin examples 301
   A string editor 303
   An external editor 309

A Miscellaneous Widget-Building Tips 315

Glossary 319

Index 335
<table>
<thead>
<tr>
<th>Figure Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>The header file and source file for a widget class</td>
<td>5</td>
</tr>
<tr>
<td>Button widgets</td>
<td>9</td>
</tr>
<tr>
<td>A container with toggle-button children</td>
<td>10</td>
</tr>
<tr>
<td>A compound widget</td>
<td>10</td>
</tr>
<tr>
<td>Default anchoring for a container widget</td>
<td>13</td>
</tr>
<tr>
<td>Methods being chained up</td>
<td>17</td>
</tr>
<tr>
<td>Inherited methods</td>
<td>17</td>
</tr>
</tbody>
</table>
What you’ll find in this guide

The Building Custom Widgets guide is intended for Photon programmers who want to create custom widgets.

If you’re familiar with earlier versions of Photon, you should read:

- What’s new in Photon for QNX Neutrino 6.3.0 Service Pack 1
- What’s new in Photon for QNX Neutrino 6.3
- What’s new in Photon for QNX Neutrino 6.2.1
- What’s new in Photon for QNX Neutrino 6.2.0

to find out what has changed in this version.

This manual contains everything you need to know about building your own Photon widgets. Sample code is used throughout this manual to demonstrate new concepts.

<table>
<thead>
<tr>
<th>When you want to:</th>
<th>Go to:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Learn about widget concepts, attributes, and behavior</td>
<td>Overview</td>
</tr>
<tr>
<td>Initialize a widget class, set up a widget instance structure, or set application resources</td>
<td>Life Cycle of a Widget</td>
</tr>
<tr>
<td>Define a widget class, its resources, methods, or actions</td>
<td>Anatomy of a Widget</td>
</tr>
<tr>
<td>Choose a widget superclass to base your custom widget on</td>
<td>Using Widget Superclasses</td>
</tr>
<tr>
<td>Customize a list widget</td>
<td>Creating a List Widget</td>
</tr>
<tr>
<td>Customize a tree widget</td>
<td>Creating a Tree Widget</td>
</tr>
<tr>
<td>Add a custom widget to the PhAB widget palette</td>
<td>Binding Widgets into PhAB</td>
</tr>
<tr>
<td>Manage widgets using the supplied convenience functions</td>
<td>Widget Building Library API</td>
</tr>
<tr>
<td>Create a custom PhAB resource editor for your widget’s resources</td>
<td>Creating custom resource editors</td>
</tr>
<tr>
<td>Read about the resource editor plugin API</td>
<td>The resource editor API</td>
</tr>
</tbody>
</table>

continued...
Typographical conventions

Throughout this manual, we use certain typographical conventions to distinguish technical terms. In general, the conventions we use conform to those found in IEEE POSIX publications. The following table summarizes our conventions:

<table>
<thead>
<tr>
<th>Reference</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code examples</td>
<td>if( stream == NULL )</td>
</tr>
<tr>
<td>Command options</td>
<td>-lR</td>
</tr>
<tr>
<td>Commands</td>
<td>make</td>
</tr>
<tr>
<td>Environment variables</td>
<td>PATH</td>
</tr>
<tr>
<td>File and pathnames</td>
<td>/dev/null</td>
</tr>
<tr>
<td>Function names</td>
<td>exit()</td>
</tr>
<tr>
<td>Keyboard chords</td>
<td>Ctrl-Alt-Delete</td>
</tr>
<tr>
<td>Keyboard input</td>
<td>something you type</td>
</tr>
<tr>
<td>Keyboard keys</td>
<td>Enter</td>
</tr>
<tr>
<td>Program output</td>
<td>login:</td>
</tr>
<tr>
<td>Programming constants</td>
<td>NULL</td>
</tr>
<tr>
<td>Programming data types</td>
<td>unsigned short</td>
</tr>
<tr>
<td>Programming literals</td>
<td>0xFF, &quot;message string&quot;</td>
</tr>
<tr>
<td>Variable names</td>
<td>stdin</td>
</tr>
<tr>
<td>User-interface components</td>
<td>Cancel</td>
</tr>
</tbody>
</table>

We use an arrow (→) in directions for accessing menu items, like this:

You’ll find the Other... menu item under Perspective→Show View.

We use notes, cautions, and warnings to highlight important messages:
Notes point out something important or useful.

CAUTION: Cautions tell you about commands or procedures that may have unwanted or undesirable side effects.

WARNING: Warnings tell you about commands or procedures that could be dangerous to your files, your hardware, or even yourself.

Note to Windows users

In our documentation, we use a forward slash (/) as a delimiter in all pathnames, including those pointing to Windows files.

We also generally follow POSIX/UNIX filesystem conventions.

Technical support

To obtain technical support for any QNX product, visit the Support + Services area on our website (www.qnx.com). You’ll find a wide range of support options, including community forums.

What’s new in Photon for QNX Neutrino 6.3.0 Service Pack 1

Two new functions:

- PtGetCallbackList()
- PtInvokeCallbackType()

What’s new in Photon for QNX Neutrino 6.3

Resource editor plugin API

The chapters dealing with the resource editor plugin API are new. These chapters are:

- Creating custom resource editors
- The resource editor API
- Resource editor plugin example

The new function PtFindResource() replaces PtFindResourceRecord(), which is deprecated.

Binding Widgets into PhAB — There is a new ushort resource type.
What’s new in Photon for QNX Neutrino 6.2.1

Errata

$PtWidgetAbove()$ Corrected the description.

What’s new in Photon for QNX Neutrino 6.2.0

Overview

- Anchoring is now done by $PtWidget$ instead of $PtContainer$.

Anatomy of a Widget

- The section on $PtResourceRec_t$ resource records includes new $mod_f$ special values ($Pt\_CHANGE\_CANVAS$ and $Pt\_CHANGE\_CANVAS\_REDRAW$) and a new $arg_value$ type ($Pt\_ARG\_IS\_IMAGE$).

- Widget classes now include a textual description. See “Widget class resource table.”

- $PtContainerChildRedirect()$ no longer exists; use the $Pt\_SET\_CHILD\_REDIRECT\_F$ manifest in your widget’s class-creation function instead. See “Container widget anatomy.”

Creating a List Widget

- The prototypes have changed for $PtSuperClassGenListMouse()$ and $PtSuperClassGenListSelect()$.

Creating a Tree Widget

- When expanding an item, the Tree Item State method returns zero to permit the expansion, or a nonzero value to prevent it — $PtGenTreeExpand()$ returns this value. When collapsing an item, the Tree Item State method returns a nonzero value if the item is destroyed, or zero if the item still exists — $PtGenTreeCollapse()$ returns this value (the prototype has changed).

- $PtGenTreeAddAfter()$ and $PtGenTreeAddFirst()$ now return 0 on success, or -1 if the item is already in a tree.

- $PtGenTreeFreeItems()$ now returns 0 on success, or -1 if the items are still in a tree.

Binding Widgets into PhAB

The method for binding widgets into PhAB has changed. If PhAB can open the shared object for your widget, it can display your widget correctly, and you can work with them “live.”

Widget Building Library API

New functions:

- $PtAnchorDeregister()$ — replaces $PtContainerDeregister()$. 
- *PtAnchorRegister()* — replaces *PtContainerRegister().*

- *PtResizeCanvas()* — similar to *PtAttemptResize(),* but easier to use.

- *PtContainerChildRedirect()* no longer exists; use the
  Pt_SET_CHILD_REDIRECT_F manifest in your widget’s class-creation function
  instead. See “Container widget anatomy” in the Anatomy of a Widget chapter.

- *PtGetAnchoredExtent()* no longer exists; anchors are handled by the widget library.

- *PtInvokeResizeCallbacks()* is now a `void` function. It assumes that the widget
  passed to it is a container.

**Miscellaneous Widget-Building Tips**

Various tips have been added.
In this chapter…

Subclassing widgets 3
Design considerations 4
A quick look at a widget 5
Types of widgets 8
When you build an application, you typically use the standard widgets provided with the Photon widget library. But occasionally you’ll have a reason for not using these standard widgets.

For example, Photon widgets require a fair amount of memory, so if you’re drawing a diagram having several `PtRect`, `PtLine` and `PtPolygon` widgets, it might be better to create a single widget that can draw the entire diagram using `Pg*()` graphic primitives.

Another reason for creating a custom widget is the need for a user interface object whose features aren’t supported by standard widgets. In this case, you have three options:

- Code the required user interface functionality directly into your application.
- Use the raw widget, `PtRaw` — see the Raw Drawing and Animation chapter of the Photon Programmer’s Guide.
- Build a custom widget.

Placing custom user-interface code within a widget has its advantages:

- The widget becomes a self-contained module that can be used within other applications.
- The widget can take advantage of the Photon widget engine, which has several features available only to widgets (e.g. automatic routing of events to the appropriate widgets, which allows widgets to automatically redraw and repair themselves when required).
- The user-interface code is separated from the application-specific code (the primary goal of widgets).

The process of building a widget in Photon is very straightforward. Because all widgets follow a specific structure or layout, you can start with a standard widget template or an existing widget that has similar functionality.

Subclassing widgets

Building a custom widget is also known as subclassing a widget because widgets are always built as descendants of existing widget superclasses. Subclassing lets you enhance or modify the behavior associated with an existing widget superclass, provided you understand how the superclass is implemented. The Using Widget Superclasses chapter describes the most common widget superclasses.

When creating a custom widget, you should subclass it under `PtBasic` or one of `PtBasic`’s subclasses. The `PtBasic` superclass is the ancestral widget class of most standard Photon widgets. Subclassing a widget under `PtBasic` lets the widget inherit the features common to most Photon widgets, such as support for keyboard traversal, highlighting, and focus. If you create a widget that doesn’t support these common features, it may look out of place in a Photon application.
CAUTION:
If you’ll be adding a custom widget to the Photon Application Builder (PhAB) widget palette, there are a few special design considerations you must be aware of. Before you begin designing your widget, read the Binding Widgets into PhAB chapter.

Design considerations

Part of building a custom widget involves deciding what it will look like and how the user will interact with it. You’ll need to decide on the resources that control the widget and pick an appropriate widget superclass to start with.

Appearance
Designing the look of a widget involves an exercise in visualization. The widget’s appearance could depend on whether a user will interact with the widget. You should also consider whether a developer may need to customize your widget. Once you’ve decided on the look, you can start designing the widget’s attributes.

Attributes
Widget attributes are defined as resources within your widget. The resources should be all things that the developer can customize. The resources should also define all the callbacks your widget will generate.

When deciding on these attributes, consider how to represent them in the widget’s instance structure. For example, will the attribute be a string, a structure, or a pointer to an allocated array? Each of these attributes (whether they are read, write, or read/write) will be your widget’s resources.

Widget actions
If your widget will be interactive, you’ll have to determine which Photon events your widget will be sensitive to. These events must be set up in the class-creation function.

You can have all events go through the same function, or you can define a function for each event type. The functions that handle events should read the event data and take the appropriate action. For example, a click on your widget might change its internal state. If appropriate, your widget may need to be damaged so a subsequent redraw operation can reflect the widget’s new state.

Superclass
More often than not, some or most of the resources your widget requires are already available within an existing widget superclass. When you build your widget as a subclass of a superclass, much of your widget’s behavior will be inherited from the superclass.
A quick look at a widget

A widget is usually made up of two files:

- a header file, which defines the widget’s instance structure and resource declarations needed by applications to use the widget
- a source code file, which contains the widget’s class structure and the actual code to implement the widget. This code includes methods, class-level functions that define how the widget initializes itself, draws itself, calculates its extent, and so on.

The following diagram depicts the header and source code files for a simple example of a widget, `ShadowedBox`.

The header file and source file for a widget class.

This sample widget will draw a shadowed box, define a resource for changing the color of the shadow, and define a resource for specifying the offset of the shadow.

This sample widget will be subclassed as follows:

```
PtWidget → PtBasic → ShadowedBox
```

Since `ShadowedBox` will be a subclass of `PtBasic`, it will inherit all the resources and behavior of this standard Photon widget.

### Widget header file

Photon widgets comprise two structures: the class structure and the instance structure. The class structure is defined in the source file and is discussed in the next section. The instance structure is contained in the header file, as shown below in an excerpt from the `ShadowedBox.h` header file:

```c
/* ShadowedBox.h - widget header file */
#include <Pt.h>
/* widget resources */
#define SBW_SHADOW_COLOR Pt_RESOURCE( Pt_USER( 0 ), 0 )
#define SBW_SHADOW_OFFSET Pt_RESOURCE( Pt_USER( 0 ), 1 )
```
/* widget instance structure */
typedef struct shadowed_box_widget{
    PtBasicWidget_t basic;
      PgColor_t shadow_color;
       short shadow_offset;
} ShadowedBoxWidget;

/* widget class pointer */
extern PtWidgetClassRef_t *ShadowedBox;

Instance structure members

Each instance of a widget has its own copy of the instance structure, which carries the complete widget state including everything that can be unique to that instance. Widgets define public resource manifests, which correspond to the members of the instance structure.

The first member of the instance structure must be the instance structure of the widget’s immediate superclass. ShadowedBox is a subclass of PtBasic, so the first member of its instance structure is of type PtBasicWidget_t.

When a widget class is created, the members of the instance structure are given default values or can be set using resource manifests. The instance structure members can be changed by calling PtSetResources() and read by calling PtGetResources().

The header file also defines an external reference to the widget class pointer.

Widget source file

The organization of the source file is quite simple. First, it defines the class structure. There’s only one copy of the class structure, which all widget instances use. It’s initialized through the class-creation function when the first instance of this class is created.

Class structure members

The class structure’s members contain pointers to methods and resources that control how the Photon library handles instances of this class.

The following class structure is found in the ShadowedBox.c source file:

/* ShadowedBox.c - widget source file */

#include "ShadowedBox.h"

/* prototype declarations */
PtWidgetClass_t *CreateShadowedBoxClass( void );

/* widget class pointer - class structure, create function */
PtWidgetClassRef_t WShadowedBox = { NULL,
   CreateShadowedBoxClass };
PtWidgetClassRef_t *ShadowedBox = &WShadowedBox;

//
// ShadowedBox defaults function
// static void shadowedbox_dflts( PtWidget_t *widget )
{
    ShadowedBoxWidget *sb = ( ShadowedBoxWidget * ) widget;
    PtBasicWidget_t *basic = ( PtBasicWidget_t * ) widget;

    basic->fill_color = Pg_WHITE;
    sb->shadow_color = Pg_BLACK;
    sb->shadow_offset = 4;
}

// ShadowedBox draw function
//
static void shadowedbox_draw( PtWidget_t *widget, PhTile_t *damage )
{
    ShadowedBoxWidget *sb = ( ShadowedBoxWidget * ) widget;
    PtBasicWidget_t *basic = ( PtBasicWidget_t * ) widget;

    PhRect_t shadow_rect, rect;
    PgColor_t color;

    // We want to use basic's draw function to get borders
    // and default focus rendering... but we don't want it to
    // fill the background with the basic fill color,
    // so we set the fill color to transparent for Basic's draw func.
    color = basic->fill_color;
    basic->fill_color = Pg_TRANSPARENT;
    PtSuperClassDraw( PtBasic, widget, damage );

    // we don't want to draw outside our canvas! So we clip.
    PtCalcCanvas( widget, &rect );
    PtClipAdd( widget, &rect );
    basic->fill_color = color;
    shadow_rect = rect;
    shadow_rect.ul.x += sb->shadow_offset;
    shadow_rect.ul.y += sb->shadow_offset;
    PgSetStrokeColor( sb->shadow_color );
    PgSetFillColor( sb->shadow_color );
    PgDrawRect( &shadow_rect, Pg_DRAW_FILL_STROKE );
    PgSetFillTransPat( basic->trans_pattern );
    PgSetStrokeColor( basic->color );
    PgSetFillColor( color );
    rect.lr.x -= sb->shadow_offset;
    rect.lr.y -= sb->shadow_offset;
    PgDrawRect( &rect, Pg_DRAW_FILL_STROKE );

    /* remove the clipping */
    PtClipRemove();
}

// ShadowedBox class creation function
//
PtWidgetClass_t *CreateShadowedBoxClass( void )
{
    // define our resources
    static PtResourceRec_t resources[] = {
        SBW_SHADOW_COLOR, Pt_CHANGE_REDRAW, 0,
        Pt_ARG_IS_NUMBER( ShadowedBoxWidget, shadow_color ), 0,
        SBW_SHADOW_OFFSET, Pt_CHANGE_RESIZE_REDRAW, 0,
        Pt_ARG_IS_NUMBER( ShadowedBoxWidget, shadow_offset ), 0,
        }
In the source file above, the class definition defines the `shadowedbox_dflts()` and `shadowedbox_draw()` functions for the Defaults and Draw methods. It also defines how the `SBW_SHADOW_COLOR` and `SBW_SHADOW_OFFSET` resources are handled.

As you can see from this example, creating a simple widget doesn’t take a lot of code.

Let’s review the fundamental structure of a widget source code file:

<table>
<thead>
<tr>
<th>Section</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header files</td>
<td><code>#include &quot;ShadowedBox.h&quot;</code></td>
</tr>
<tr>
<td>Class pointer declaration</td>
<td><code>PtWidgetClassRef_t *ShadowedBox</code></td>
</tr>
<tr>
<td>Method functions</td>
<td><code>shadowedbox_dflts(), shadowedbox_draw()</code></td>
</tr>
<tr>
<td>Class-creation function</td>
<td><code>CreateShadowedBoxClass()</code></td>
</tr>
</tbody>
</table>

**Types of widgets**

The Photon library supplies a number of widget classes that can be used as the basis for custom widgets. To make this easy to understand, we’ve categorized widgets into three general types:

- Basic
- Container
- Compound
Class hierarchy

All Photon widgets are based on the PtWidget core widget class. It defines the characteristics common to all standard widgets, such as position and dimension. Even if your widget is different from any other widget, it will still inherit the features of the PtWidget widget class. Most custom widgets, however, will be subclassed under one of the other widget superclasses.

PtWidget also supports anchoring features that allow widgets to be anchored to their parent.

Basic widgets

Basic widgets are single-entity objects. They don’t contain other widgets and aren’t built from other widgets. An example would be a button widget.

You can’t put other widgets inside a button, and the button itself isn’t built from other widgets. Our sample ShadowedBox widget is a basic widget.

Although we call them “basic” widgets, this is only a classification and doesn’t imply that they can’t be powerful. In fact, one of the most complex widgets in the Photon library, PtText, is a basic widget.

Container widgets

Container widgets can have other widgets as children. Containers have the option of managing the geometry of their children (e.g. PtGroup) or leaving the widgets as they are (e.g. PtContainer).

The PtContainer class extends the PtBasic class definition to include many new methods for controlling and reacting to the widget children placed inside the container. Using these new methods, a container could, for example, detect when a new child is added and automatically resize itself to accommodate the new widget.

Widget children can be selectively blocked by a child-redirector function. This is useful when you want to create a widget that accepts only specific widgets as children (e.g. PtMenuBar accepts only PtMenuButton widgets as children). All other widgets are redirected to the container’s parent widget. This causes them to be created at the same level as the container instead of as a child of the container.
Compound widgets

Compound widgets are built from exported subordinate widgets. The exporting mechanism allows access to the underlying widgets using their resources and convenience functions. This reduces the need to duplicate every subordinate widget resource in the compound widget’s resource list.

You have to set up resources only for new functionality or for resolving conflicts when more than one of the same widget class has been exported. A blocking mechanism lets compound widgets block access to any of the subordinate resources. This lets you prevent any actions that would adversely affect the look and behavior of the compound widget.

Since the `PtCompound` class is a descendant of `PtContainer`, it inherits all the functionality of a container widget. This means compound widgets can support child widgets other than the ones they’re built from. Since this feature isn’t usually desired, you can use a compound widget’s child-redirector function to automatically reject the addition of unwanted widgets.

A good example of a compound widget is `PtComboBox` — it’s built using the `PtText` and `PtList` widget classes and adds its own functionality.

Another good example is `PtDivider` — it’s built using a `PtGroup` widget; separator bars are added automatically between the widgets in the group.
Chapter 2
Life Cycle of a Widget

In this chapter...

All widgets 13
Basic widgets 13
Container widgets 21
Compound widgets 22
This chapter examines the life cycle of a basic widget using the ShadowedBox widget as an example. We’ll show you how to create, use, and destroy this widget. We’ll also outline the life cycle differences between basic, container, and compound widgets.

All widgets

If a widget is already realized or in the process of being realized, it’s automatically anchored by the widget library.

```
Parent container

Child container

Anchoring
```

*Default anchoring for a container widget.*

PtAnchorWidget() and PtApplyAnchors() in the widget-building library allow anchoring to be applied at any time, provided the parent widget’s extent is valid.

Basic widgets

If we were building an application and wanted to use the custom ShadowedBox widget (a “basic” widget), we would simply create it by calling PtCreateWidget():

```c
PtWidget_t *box;
PtArea_t area = { 10, 10, 100, 100 };
PtArg_t args[2];
PtSetArg( &args[0], Pt_ARG_AREA, &area, 0 );
PtSetArg( &args[1], SBW_SHADOW_COLOR, Pg_BLUE, 0 );
box = PtCreateWidget( ShadowedBox, Pt_DEFAULT_PARENT, 2, args );
PtRealizeWidget( box );
```

What happens when we call PtCreateWidget()? What does the widget library do? Let’s follow through the widget creation process and examine the various parts of the widget code as they get executed.
Instantiating the widget

The first step, instantiating the widget, creates an instance of the widget class. The process of instantiating a widget is as follows:

1. The widget class is created and initialized.
2. All members of the widget instance structure are initialized to default values.
3. The resources specified through PtCreateWidget() are applied.

Creating and initializing the widget class

When a widget is created via a call to PtCreateWidget(), the specified widget class (i.e. ShadowedBox) is first checked to see if it has been initialized. Since this is the first time this widget has been created, the widget’s initialization function is called to create the widget class.

In the ShadowedBox.c source file, the widget class is defined as a class reference structure:

```c
PtWidgetClassRef_t WShadowedBox = { NULL, 
    CreateShadowedBoxClass }; 
PtWidgetClassRef_t *ShadowedBox = &WShadowedBox; 
```

The first member is the widget class pointer, which is initialized to NULL. The second member is the widget class-creation function. If the value of the class pointer is NULL, the class-creation function is called and it sets the widget class pointer in the structure.

Let’s look at CreateShadowedBoxClass() from our example to see how it defines the ShadowedBox class:

```c
// ShadowedBox class creation function
/**
 * @brief Create the ShadowedBox class
 */
PtWidgetClass_t *CreateShadowedBoxClass( void ) 
{
    // define our resources
    static PtResourceRec_t resources[] = {
        SBW_SHADOW_COLOR, Pt_CHANGE_REDRAW, 0, 
        Pt_ARG_IS_NUMBER( ShadowedBoxWidget, shadow_color ), 0, 
        SBW_SHADOW_OFFSET, Pt_CHANGE_RESIZE_REDRAW, 0, 
        Pt_ARG_IS_NUMBER( ShadowedBoxWidget, shadow_offset ), 0, 
    };
    // set up our class member values
    static PtArg_t args[] = {
        { Pt_SET_VERSION, 110},
        { Pt_SET_STATE_LEN, sizeof( ShadowedBoxWidget ) },
        { Pt_SET_DFLTS_F, (long)shadowedbox_dflts },
        { Pt_SET_DRAW_F, (long)shadowedbox_draw },
        { Pt_SET_FLAGS, 0, Pt_RECTANGULAR },
        { Pt_SET_NUM_RESOURCES, 
            sizeof( resources ) / sizeof( resources[0] ) },
        { Pt_SET_RESOURCES, (long)resources, 
            sizeof( resources ) / sizeof( resources[0] ) },
    };
    // create the widget class
```
return( ShadowedBox->wclass = PtCreateWidgetClass(
        PtBasic, 0, sizeof( args )/sizeof( args[0] ), args ) );
}

The class definition includes:

Resources array

The static array resources defines the class resources. The class
resources are accessed through a table of resources unique to
the ShadowedBox class.

Widget class array

The static array args defines the ShadowedBox class. It defines
ShadowedBox’s class methods and its table of resources.

The args array is passed to PtCreateWidgetClass() as follows:

    return( ShadowedBox->wclass = PtCreateWidgetClass(
        PtBasic, 0, sizeof( args )/sizeof( args[0] ), args ) );

In PtCreateWidgetClass(), space is allocated for the
ShadowedBox class and its superclass’s structure (i.e.
PtBasic) is copied in. Then the args array is used to set up the
custom class and its table of unique resources.

Setting default values

After the ShadowedBox class is initialized, space is allocated for its instance
structure. The Defaults methods of all ancestors of the ShadowedBox class are called
next, ending with ShadowedBox’s Defaults method (e.g. shadowedbox_dflts()):

    // ShadowedBox defaults function
    //
    static void shadowedbox_dflts( PtWidget_t *widget )
    {
        ShadowedBoxWidget *sb = ( ShadowedBoxWidget * ) widget;
        PtBasicWidget_t *basic = ( PtBasicWidget_t * ) widget;

        basic->fill_color = Pg_WHITE;
        sb->shadow_color = Pg_BLACK;
        sb->shadow_offset = 4;
    }

To access the private members of the instance structure, this Defaults method casts the
widget pointer to a ShadowedBoxWidget pointer and a PtBasicWidget_t pointer.
It then defaults the fill_color member to Pg_WHITE, the shadow_color member to
Pg_BLACK and the shadow_offset member to 4.

Setting application resources

After the Defaults method is called, the array of resources provided to
PtCreateWidget() is applied. In the sample source code, these two resources are
passed in args:

    PtSetArg( &args[0], Pt_ARG_AREA, &area, 0 );
    PtSetArg( &args[1], SBW_SHADOW_COLOR, Pg_BLUE, 0 );
The first resource manifest, \texttt{Pt\_ARG\_AREA}, sets the widget area (position and size). Because this resource isn’t defined by the \texttt{ShadowedBox} class, it’s handled by the first superclass that defines \texttt{Pt\_ARG\_AREA} (i.e. \texttt{PtWidget}).

The second resource manifest \texttt{SBW\_SHADOW\_COLOR} is handled by the \texttt{ShadowedBox} class. Because \texttt{ShadowedBox} doesn’t define a method for setting the resource value, the Photon widget library takes care of assigning a new value to this instance structure member (i.e. the value is changed from the default, \texttt{Pg\_BLACK}, to \texttt{Pg\_BLUE}).

Finally, \texttt{PtCreateWidget()} returns the widget pointer and the application receives access to a newly created instance of the \texttt{ShadowedBox} class:

\begin{verbatim}
box = PtCreateWidget( ShadowedBox, Pt_DEFAULT_PARENT,
                2, args );
\end{verbatim}

Now \texttt{box} points to an instance of the \texttt{ShadowedBox} class, but the widget itself won’t be displayed until it’s realized.

## Realizing a widget instance

There are a number of class methods used in the process of realizing a widget. These methods are executed in a specific order that we’ll call “steps”:

<table>
<thead>
<tr>
<th>Step</th>
<th>What it does</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initialization</td>
<td>Initializes the widget instance.</td>
</tr>
<tr>
<td>Extent method</td>
<td>Calculates the actual widget extent based on the area specified.</td>
</tr>
<tr>
<td>Connection method</td>
<td>Creates any optional Photon regions.</td>
</tr>
<tr>
<td>Realization method</td>
<td>Performs any operations required immediately prior to drawing.</td>
</tr>
<tr>
<td>Draw method</td>
<td>Draws the widget.</td>
</tr>
</tbody>
</table>

The \texttt{ShadowedBox} example doesn’t define an Initialization method, an Extent method, or a Connection method. These methods are handled by \texttt{PtWidget} or \texttt{PtBasic}.

## Processing methods

Before we get into a detailed discussion of the individual methods used in a widget class, we need to look at their usage in general. Class methods can be \texttt{chained} up or down or \texttt{inherited}.

When a method is chained, the method of each class in the widget hierarchy is executed. Execution can start at the lowest class (e.g. \texttt{ShadowedBox}) in the hierarchy and go up, or at the highest class (i.e. \texttt{PtWidget}) and go down.
Some chained methods can be stopped. This means any class in the hierarchy can stop the chaining process to prevent the superclass or subclassed methods from being executed. When chaining is stopped, the class stopping the chain is responsible for supplying the equivalent functionality otherwise provided by another class.

Inherited methods work differently: one method only is called, as determined by the lowest class in the hierarchy to define it (e.g. our `ShadowedBox` class doesn’t define an Extent method, so the Extent method is inherited from the `PtBasic` superclass).

This means `PtBasic`’s Extent method is used to determine the size of the `ShadowedBox` widget. Inherited methods make building a widget class easier because only the methods unique to a class need to be defined.

Let’s look at the class methods in more detail. The following table shows whether the methods of a basic widget are chained or inherited:

<table>
<thead>
<tr>
<th>Method</th>
<th>Processing</th>
<th>Processing note</th>
</tr>
</thead>
<tbody>
<tr>
<td>Defaults</td>
<td>Chained down</td>
<td>Can’t be stopped</td>
</tr>
<tr>
<td>Initialization</td>
<td>Chained up</td>
<td>Return Pt_END to stop</td>
</tr>
<tr>
<td>Extent</td>
<td>Inherited</td>
<td></td>
</tr>
<tr>
<td>Connection</td>
<td>Chained up</td>
<td>Return Pt_END to stop</td>
</tr>
<tr>
<td>Realization</td>
<td>Inherited</td>
<td></td>
</tr>
<tr>
<td>Draw</td>
<td>Inherited</td>
<td></td>
</tr>
<tr>
<td>Unrealization</td>
<td>Chained up</td>
<td>Can’t be stopped</td>
</tr>
</tbody>
</table>

continued…
### Basic widgets

© 2007, QNX Software Systems GmbH & Co. KG.

<table>
<thead>
<tr>
<th>Method</th>
<th>Processing</th>
<th>Processing note</th>
</tr>
</thead>
<tbody>
<tr>
<td>Destruction</td>
<td>Chained up</td>
<td>Can’t be stopped</td>
</tr>
<tr>
<td>Set Resources</td>
<td>Inherited</td>
<td></td>
</tr>
<tr>
<td>Get Resources</td>
<td>Inherited</td>
<td></td>
</tr>
</tbody>
</table>

PtBasic extends these methods with three other methods. These methods are available only to subclasses of PtBasic:

<table>
<thead>
<tr>
<th>Method</th>
<th>Processing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Got Focus</td>
<td>Inherited</td>
</tr>
<tr>
<td>Lost Focus</td>
<td>Inherited</td>
</tr>
<tr>
<td>Calc Opaque Rect</td>
<td>Inherited</td>
</tr>
</tbody>
</table>

### Initialization method

This is the first class method called during the realization process. This method is chained up; the Initialization method of the ShadowedBox class would be called first, followed by the Initialization method of the PtBasic class, ending with PtWidget’s Initialization method.

### Extent method

This inherited method is used to determine the exact size of the widget based on default values and/or the widget’s position, size, margins, borders, and highlighting information, as set by the program. An Extent method isn’t defined for ShadowedBox, so the Extent method is inherited from the PtBasic superclass.

### Connection method

The Connection method is chained up. It’s used primarily for creating any Photon regions needed by a widget. Due to its simplicity, the ShadowedBox widget doesn’t require any regions — a superclass method will create regions for ShadowedBox only if needed (e.g. a custom cursor is defined).

### Realization method

The Realization method is used to adjust any region attributes prior to the actual drawing of a widget. Because ShadowedBox doesn’t define a Realization method, it’s inherited from a superclass.

### Draw method

This is the last class method called during the realization process. It’s used to render a widget on the screen. This method can be inherited, but since ShadowedBox defines
its own Draw method (shown below), it’s executed instead of a superclass’s Draw method:

```
// ShadowedBox draw function
static void shadowedbox_draw( PtWidget_t *widget, PhTile_t *damage )
{
    ShadowedBoxWidget *sb = ( ShadowedBoxWidget * ) widget;
    PtBasicWidget_t *basic = ( PtBasicWidget_t * ) widget;
    PhRect_t shadow_rect, rect;
    PgColor_t color;

    // We want to use basic’s draw function to get borders
    // and default focus rendering... but we don’t want it to
    // fill the background with the basic fill color,
    // so we set the fill color to transparent for Basic’s draw func.
    color = basic->fill_color;
    basic->fill_color = Pg_TRANSPARENT;
    PtSuperClassDraw( PtBasic, widget, damage );

    // we don’t want to draw outside our canvas! So we clip.
    PtCalcCanvas( widget, &rect );
    PtClipAdd( widget, &rect );

    basic->fill_color = color;
    shadow_rect = rect;
    shadow_rect.ul.x += sb->shadow_offset;
    shadow_rect.ul.y += sb->shadow_offset;
    PgSetStrokeColor( sb->shadow_color );
    PgSetFillColor( sb->shadow_color );
    PgDrawRect( &shadow_rect, Pg_DRAW_FILL_STROKE );

    PgSetFillTransPat( basic->trans_pattern );
    PgSetStrokeColor( basic->color );
    PgSetFillColor( color );
    rect.lr.x -= sb->shadow_offset;
    rect.lr.y -= sb->shadow_offset;
    PgDrawRect( &rect, Pg_DRAW_FILL_STROKE );

    /* remove the clipping */
    PtClipRemove();
}
```

Let’s look at the code in more detail. From the function declaration:

```
static void shadowedbox_draw( PtWidget_t *widget, PhTile_t *damage )
```

The Draw method is passed two arguments. The `widget` argument is the widget’s pointer and the `damage` argument is a list of damage rectangles to be applied to the widget when it’s drawn.

The program can use the damage list to make intelligent decisions about what to draw. However, drawing operations for most widgets are minimal and any extra processing might not be worthwhile.

Notice the widget pointer is of type `PtWidget_t`, which means the pointer must be cast to a `ShadowedBoxWidget` pointer to allow access to the `ShadowedBox` widget’s internal structure members. Since `PtBasic` is a superclass of `ShadowedBox`, we can
cast the widget pointer to a `PtBasicWidget_t` structure too; this allows access to the `basic` widget members without resorting to `sb->basic`.

```c
ShadowedBoxWidget *sb = (ShadowedBoxWidget *) widget;
PtBasicWidget_t *basic = (PtBasicWidget_t *) widget;
```

Since the Draw method isn’t chained, all drawing of the `ShadowedBox` widget, including the borders and highlighting common to all Photon-style widgets, must be done within the Draw method of the `ShadowedBox` class. For convenience, the Photon widget building library allows you to call the draw method of a superclass using `PtSuperClassDraw()` as shown below:

```c
// We want to use basic’s draw function to get borders
// and default focus rendering... but we don't want it to
// fill the background with the basic fill color,
// so we set the fill color to transparent for Basic’s draw func.
color = basic->fill_color;
basic->fill_color = Pg_TRANSPARENT;
PtSuperClassDraw( PtBasic, widget, damage );
```

After this, we add clipping so we won’t draw beyond the widget’s canvas:

```c
// we don't want to draw outside our canvas! So we clip.
PtCalcCanvas( widget, &rect );
PtClipAdd( widget, &rect );
```

The rest of the code in the Draw method determines the widget’s position and size based on its area and margins, and then draws a shadowed box using low-level Photon `Pg*()` graphics functions. Before leaving the Draw method, the clipping we added is removed by calling `PtClipRemove()`.

When the draw buffer is flushed later on, the widget appears on the screen.

### After realizing a widget

After a widget is realized, it must be ready to:

- handle any changes made to its resources
- redraw itself when damaged or exposed

Fortunately, both of these operations are handled automatically because of the way widget classes are defined — the resource list specifies what should be done if the value of a resource changes:

```c
static PtResourceRec_t resources[] = {
  SBW_SHADOW_COLOR, Pt_CHANGE_REDRAW, 0,
  Pt_ARG_IS_NUMBER( ShadowedBoxWidget, shadow_color ), 0,
  SBW_SHADOW_OFFSET, Pt_CHANGE_RESIZE_REDRAW, 0,
  Pt_ARG_IS_NUMBER( ShadowedBoxWidget, shadow_offset ), 0,
};
```

In the example source code for `ShadowedBox`, both resources are defined with a `Pt*REDRAW` manifest so they’re automatically marked as damaged. This causes a redraw if the value changes.

When a widget is damaged for any reason, the Photon library automatically invokes the widget’s Draw method to repair the damage. If any of the inherited resources are changed, they’re handled according to the resource declaration of the first superclass to define that resource.
Destroying a widget

The widget must handle its own destruction. This process involves:

- unrealizing the widget
- destroying the widget

Unrealization method

The purpose of this method is to undo everything of consequence done during realization. The Unrealization method is responsible for:

- removing any open regions owned by a widget
- erasing the widget from the screen through the widget library
- making the widget noninteractive

This method is chained up. Since the `ShadowedBox` class doesn’t define an Unrealization method, `PtBasic`’s Unrealization method will be called, followed by `PtWidget`’s Unrealization method.

Destruction method

This method is responsible for releasing any system resources used by a widget but not referenced by a widget resource. Like the Unrealization method, it’s chained up. This lets all classes in the hierarchy release system resources in turn.

Other methods

The remaining class methods aren’t described in this section, but you should now have a general understanding of the overall process. The Anatomy of a Widget chapter describes these methods in detail. Let’s now look at some of the additional requirements for container and compound classes.

Container widgets

The life cycle of a container widget is similar to that of a basic widget. The main difference is that container widgets must support widget children, including the following features:

- child constraints
- child redirection

The Extent method of a container class is very important — it must enforce its resize policy based on its children. This means a container widget may need to determine its size based on the positions and sizes of its children. For this reason, all child widgets are extented before the container is extented.
Child constraints

The container class extends the PtBasic class definition with a number of child-constraint methods. These methods let a container adjust itself automatically according to any changes made to its children. These child-constraint methods include:

- Child Created
- Child Realized
- Child Moved/Resized
- Child Unrealized
- Child Destroyed
- Child Setting Resources
- Child Getting Resources
- Child Getting Focus
- Child Losing Focus

A container dynamically identifies which child constraints need to be handled by setting and clearing bits in the flags member of its instance of the class structure. For example:

```c
    cntr->flags |= Pt_CHILD_CREATED;
```

Any child-constraint method a container will support must have its bit set in `cntr->flags` or the method won’t be invoked.

Child redirection

A custom container class can optionally provide a child-redirector function to restrict certain widget classes from being added to the container. This function allows the container to redirect parentage to other widgets inside or outside the container itself. The redirector function is usually defined in the class-creation function. For example:

```c
    { Pt_SET_CHILD_REDIRECT_F, (long)PtSampContainerRedirect },
```

The PtMenuBar widget uses this feature to prevent certain types of widgets from being added to a menu bar. For more information, see “Container widget anatomy” in the Anatomy of a Widget chapter.

Compound widgets

The compound class is used to design widgets made from other widgets. The compound class is a subclass of the container class, so it inherits the functionality described above for containers. In addition, the compound class adds a mechanism for exporting subordinate widgets. This is achieved by extending the container class definition to include:
• the number of subordinates
• a list of subordinates
• the number of blocked resources
• a list of blocked resources

The exporting mechanism lets users set and get the resources of subordinate widgets by treating the compound widget as if it actually were the subordinate widget.

A good example is the PtComboBox widget — it’s built from a text widget and a list widget. It also defines its own behavior by supplying a pull-down button beside the text widget for accessing the list widget.

You can change the subordinate widgets inside a compound widget using the resources defined for the subordinate widgets. For example, you could change the PtText widget inside a PtComboBox widget by using the widget pointer of the PtCompound widget as follows:

```c
PtArg_t args[1];
PtSetArg( &args[0], Pt_ARG_TEXT_STRING,
          "Subordinate Text" 0 );
PtSetResources( mycombobox_widget_pointer, 1, args );
```

This is a very powerful feature that greatly simplifies the construction of compound widgets since resources don’t have to be duplicated.

However, changing the resources of subordinate widgets could destroy the look and feel of a compound widget. For this reason, we recommend that you consider using the blocking mechanism in a custom compound class. The blocking mechanism lets the compound class discard attempts to set specific resources on its subordinate widgets. The compound class can override any resource defined by its superclasses or by its subordinate widgets.

**Exporting subordinate widgets**

A compound widget identifies the widgets it intends to export in the class-creation function. This is done by setting values for the number of subordinates and the list of subordinates.

Each widget specified in the list of subordinates must be created in the Defaults method of the compound class.

For more information, see “Compound widget anatomy” in the Anatomy of a Widget chapter.
Chapter 3
Anatomy of a Widget

In this chapter...

Defining resources 27
Defining the widget class 36
Class methods 42
Widget actions 54
Container widget anatomy 56
Compound widget anatomy 62
The first part of this chapter discusses requirements applicable to all widget classes:

<table>
<thead>
<tr>
<th>This section</th>
<th>Describes:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Defining resources</td>
<td>All aspects of defining resources</td>
</tr>
<tr>
<td>Defining the widget class</td>
<td>The widget class structure and each of its fields</td>
</tr>
<tr>
<td>Class methods</td>
<td>Every class method and how each is used</td>
</tr>
<tr>
<td>Widget actions</td>
<td>How to make your widget interactive</td>
</tr>
</tbody>
</table>

You’ll find information about the additional requirements for building container and compound widgets at the end of the chapter.

## Defining resources

There are two steps to defining the resources of your widget class (aside from deciding what they should be in the first place):

1. Defining the resource manifests and numbers.
2. Defining the resource records to provide access to the resources of the widget class via resource manifests.

### Resource manifests

Define the resources to introduce for a widget class (i.e. the resources that don’t already exist in any of the widget superclasses). This is done in the widget’s header file (e.g. PtButton.h) as a series of `#define` statements:

```c
/*
 * PtButton public
 */
extern PtWidgetClassRef_t *PtButton;

#define Pt_BUTTON_ID 6

/* Resources */
#define Pt_ARG_ARM_COLOR Pt_RESOURCE( 6, 0 )
#define Pt_ARG_ARM_IMAGE Pt_RESOURCE( 6, 1 )
#define Pt_ARG_ARM_DATA Pt_ARG_ARM_IMAGE
#define Pt_ARG_ARM_FILL Pt_ARG_ARM_FILL

/*
 * PtButton private
 */

/* Widget structure */
typedef struct Pt_button_widget {
    PtLabelWidget_t label;
    PgColor_t arm_color;
    PhImage_t *arm_data;
    PhImage_t *unarmed_data;
};
```
defining resources

uchar_t arm_fill;
PtCallbackList_t *activate;
} PtButtonWidget_t;

All resource manifest names and numbers must be unique. In the above example, Pt_ARG_ARM_COLOR represents a unique resource manifest name, Pt_RESOURCE(6, 0) represents a unique resource number, and the resource itself has a corresponding entry in the widget instance structure called arm_color.

When you build a new widget, you need to pick a unique widget number for it. The header file PtT.h contains two macros to help you do this:

Pt_RESOURCE( widget_number, resource_number)
Pt_USER( widget_number )

The Pt_USER() macro is designed for widgets being created for in-house use only. If you intend to distribute the widgets you create as a public domain or commercial library, please contact QSSL Customer Service. They'll give you a unique range of widget numbers and assign your widget set a prefix. This will prevent your widget library from conflicting with another third party’s commercial widget library.

In your first in-house widget, use:

#define MY_1ST_WIDGET_RESOURCE1 Pt_RESOURCE( Pt_USER( 1 ), 0 );
#define MY_1ST_WIDGET_RESOURCE2 Pt_RESOURCE( Pt_USER( 1 ), 1 );

In your second in-house widget, use:

#define MY_2ND_WIDGET_RESOURCE1 Pt_RESOURCE( Pt_USER( 2 ), 0 );
#define MY_2ND_WIDGET_RESOURCE2 Pt_RESOURCE( Pt_USER( 2 ), 1 );

The macro Pt_USER( 1 ) defines a widget number that allows up to 1000 resources: it specifies the widget number 5,001,000. The macro Pt_USER( 2 ) defines the number 5,002,000, and so on.

The second part of the Pt_Resource() macro defines the resource number:
Pt_Resource( Pt_USER( 2 ), 5 ) defines the resource number 5,002,005. The widget defined by Pt_USER( 2 ) can define unique resources numbers from 5,002,000 to 5,002,999.

PtResourceRec_t resource records

Resource records are used to connect the resource manifests described above with the widget’s structure members. The resource records are defined as a table in the source code file and are passed as an argument when you first create the widget class.

This is a table of PtResourceRec_t items. This structure is declared as follows:

typedef struct Pt_resource_rec {
  unsigned long type;
  void (*mod_f)( PtWidget_t *, PtArg_t const *,
                 struct Pt_resource_rec const * );
  int (*query_f)( PtWidget_t const *,
                 PtArg_t *, struct Pt_resource_rec const * );
  unsigned long arg_value;
} PtResourceRec_t;
The value (manifest) to which this record is connected through a widget instance member. For example, Pt_ARG_FLAGS.

This member recognizes several special convenience values other than the address of a function. Special values include:

**Pt_CHANGE_INVISIBLE**
Set the widget member but otherwise leave the widget unaffected.

**Pt_CHANGE_REDRAW**
Change the widget member and damage the widget (this causes a redraw).

**Pt_CHANGE_INTERIOR**
Same as Pt_CHANGE_REDRAW, except the widget’s canvas alone is damaged (the borders aren’t redrawn).

**Pt_CHANGE_RESIZE**
Change the widget structure member and flag the widget for resize. The resize is held off until the end of the *PtSetResources()* call in case other Pt_CHANGE_RESIZE-type resources are set in the same call. The resize is performed via *PtMoveResizeWidget()*, and the Extent method of the widget class is called.

**Pt_CHANGE_RESIZE_REDRAW**
Same as Pt_CHANGE_RESIZE, but forces a redraw even if the widget’s dimensions or extent aren’t changed.

**Pt_CHANGE_PREVENT**
Don’t change the widget structure member and don’t affect the widget. Indicates a read-only resource.

**Pt_CHANGE_CANVAS**
Invalidate the widget’s canvas so that it’s recalculated the next time it’s requested. Additionally, resize the widget.
Pt_CHANGE_CANVAS_REDRAW

The same as Pt_CHANGE_CANVAS, but force the widget to be redrawn even if its canvas and extent aren’t changed.

query_f member

The function to call when this resource is being queried via PtGetResources(). You can provide your own function if you want to do additional processing. The third argument is a pointer to a PtResourceRec_t with the data filled in by the library which your query_f function can pass to PtGetStruct() to get the resource.

If no function is provided (i.e. query_f is NULL), the resource is reported in the normal manner (see PtSetArg() and PtGetResources() in the Photon Library Reference).

The special values of the query_f field member include:

Pt_QUERY_PREVENT

Prevents access to the resource. Any pointers provided are set to NULL. Indicates a write-only resource.

arg_value and arg_len members

These are bit-encoded fields that set members of a widget structure. The arg_value member is used for all resources; arg_len is used to set a second widget structure member. For an array, arg_len has the type and offset of the array counter. For a Boolean value, arg_len is a bitmask. Unless the resource is an array or Boolean type, arg_len is normally 0.

The data encoded into these fields includes:

• the data type of the member this resource is associated with
• the offset from the address of the widget structure to the member to get/set
• if necessary for struct or array resources, the size of the structure member.

The following macros make using arg_value and arg_len more convenient:

• Pt_ARG_IS_NUMBER
• Pt_ARG_IS_FLAGS
• Pt_ARG_IS_STRING
• Pt_ARG_IS_STRUCT
• Pt_ARG_IS_POINTER
• Pt_ARG_IS_ALLOC
• Pt_ARG_IS_LINK
• Pt_ARG_IS_CALLBACK_LIST (or Pt_ARG_IS_CALLBACK)
The sections that follow describe the values of the \textit{arg\_value} and \textit{arg\_len} members for each type of resource. For most resources, \textit{arg\_len} isn’t used; unless mentioned otherwise below, set it to 0.

Memory for some of the resources (indicated below) is allocated and freed as needed. If you have a Destruction method, you don’t need to free the memory for these resources. If you do free any memory, you \textit{must} set the pointers freed to NULL or unexpected results may occur.

### Scalar resources

<table>
<thead>
<tr>
<th>\textit{arg_value}</th>
<th>C type of \textit{member1}</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{Pt_ARG_IS_NUMBER(wgt, member1)}</td>
<td>\texttt{char, short, or long (signed or unsigned)}</td>
</tr>
</tbody>
</table>

### Flags resources

<table>
<thead>
<tr>
<th>\textit{arg_value}</th>
<th>C type of \textit{member1}</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{Pt_ARG_IS_FLAGS(wgt, member1)}</td>
<td>\texttt{char, short, or long (preferably unsigned)}</td>
</tr>
</tbody>
</table>

For Flags resources, the “mask” isn’t part of the resource — it’s just a way of telling the resource-setting API which bits the application wants to preserve.

### String resources

<table>
<thead>
<tr>
<th>\textit{arg_value}</th>
<th>C type of \textit{member1}</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{Pt_ARG_IS_STRING(wgt, member1)}</td>
<td>\texttt{char *}</td>
</tr>
</tbody>
</table>

This resource is allocated, based on the value returned by \texttt{strlen()}; see the note above.

### Struct resources
These are widget members that are of a fixed size. When setting such resources, you pass a pointer to the value, and the value is copied into the widget structure. This type is useful for structures, as well as other data types that won’t fit into a `long` (such as `float` or `double`).

**Pointer resources**

<table>
<thead>
<tr>
<th><code>arg_value</code></th>
<th>C type of <code>member1</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Pt_ARG_IS_POINTER(<code>wgt</code>, <code>member1</code>)</td>
<td>Any type of pointer, including <code>void *</code></td>
</tr>
</tbody>
</table>

The widget does a shallow copy of the pointer’s value.

**Alloc resources**

<table>
<thead>
<tr>
<th><code>arg_value</code></th>
<th>C type of <code>member1</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Pt_ARG_IS_ALLOC(<code>wgt</code>, <code>member1</code>)</td>
<td>Any type of pointer, including <code>void *</code></td>
</tr>
</tbody>
</table>

Space is allocated for the resource, with the size specified by the application; see the note above.

**Link resources**

<table>
<thead>
<tr>
<th><code>arg_value</code></th>
<th>C type of <code>member1</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Pt_ARG_IS_LINK(<code>wgt</code>, <code>member1</code>)</td>
<td>A pointer to a structure</td>
</tr>
</tbody>
</table>

The structure must start with a “next” pointer. Space is allocated for the resource; see the note above.

**Callback resources**

<table>
<thead>
<tr>
<th><code>arg_value</code></th>
<th>C type of <code>member1</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Pt_ARG_IS_CALLBACK_LIST(<code>wgt</code>, <code>member1</code>)</td>
<td>A pointer to a structure</td>
</tr>
</tbody>
</table>

The structure must start with a “next” pointer. Space is allocated for the resource; see the note above.
Boolean resources

<table>
<thead>
<tr>
<th>arg_value</th>
<th>C type of member1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pt_ARG_IS_BOOLEAN(wgt, member1)</td>
<td>char, short, int, or long (preferably unsigned)</td>
</tr>
</tbody>
</table>

The arg_len is the bitmask. It’s not stored anywhere in the widget — it’s just a constant that determines in which bit of the widget structure the resource is stored.

Array resources

<table>
<thead>
<tr>
<th>arg_value</th>
<th>C type of member1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pt_ARG_IS_ARRAY(wgt, member1)</td>
<td>A pointer to some type</td>
</tr>
</tbody>
</table>

This type of resource also uses arg_len:

<table>
<thead>
<tr>
<th>arg_len</th>
<th>C type of member2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pt_ARG_IS_NUMBER(wgt, member2)</td>
<td>char, short, or long</td>
</tr>
</tbody>
</table>

The size of each array element is the size of the type pointed to by member1. Space is allocated for the resource; see the note above.

Image resources

<table>
<thead>
<tr>
<th>arg_value</th>
<th>C type of member1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pt_ARG_IS_IMAGE(wgt, member2)</td>
<td>PhImage_t *</td>
</tr>
</tbody>
</table>

Space is allocated for the resource; see the note above. For more information about the PhImage_t structure, see the Photon Library Reference.

Examples

Now let’s look at some sample resource declarations. First let’s look back at our original widget example, the ShadowedBox widget. It defines two resources in the header file:

```c
/* widget resources */
#define SBW_SHADOW_COLOR Pt_RESOURCE( Pt_USER( 0 ), 0 )
#define SBW_SHADOW_OFFSET Pt_RESOURCE( Pt_USER( 0 ), 1 )

/* widget instance structure */
typedef struct shadowed_box_widget{
    PtBasicWidget_t  basic;
```
The source code file defines the table of resources, which connects the resources to the widget class:

```c
static PtResourceRec_t resources[] = {
    SBW_SHADOW_COLOR, Pt_CHANGE_REDRAW, 0,
    Pt_ARG_IS_NUMBER( ShadowedBoxWidget, shadow_color ), 0,
    SBW_SHADOW_OFFSET, Pt_CHANGE_RESIZE_REDRAW, 0,
    Pt_ARG_IS_NUMBER( ShadowedBoxWidget, shadow_offset ), 0,
};
```

Let's examine the first resource in the table in more detail:

- **SBW_SHADOW_COLOR** is the *type* member, which we defined as a unique resource number at the top of the widget code.

- **Pt_CHANGE_REDRAW** is the *mod_f* member, which tells the library how this resource should be handled if the resource is set. It could be either a special value (as in the example) or a function within the source file code. In the example, **Pt_CHANGE_REDRAW** tells the library that the widget needs to be redrawn if the resource value changes.

- The next member, **0**, is the *query_f* member, which tells the library how this resource should be retrieved. It can be one of:
  - **0**, which causes the widget library to retrieve the resource based on the information provided by the *arg_value* and *arg_len* bit fields
  - a function within the source file
  - **Pt_QUERY_PREVENT**

- **The next member is arg_value**. This is a bit-encoded field. In our example, **Pt_ARG_IS_NUMBER** indicates that the widget structure member associated with this resource is a value of type **char**, **short**, or **long**.

- **The last member, arg_len**, is **0** (not used).

Now let's look at a more detailed example. In the widget header file we have:

```c
/* widget resources */
#define MW_ARG_MY_CHARACTER Pt_RESOURCE( Pt_USER(1), 0 )
#define MW_ARG_MY_STRING Pt_RESOURCE( Pt_USER(1), 1 )
#define MW_ARG_MY_SHORT Pt_RESOURCE( Pt_USER(1), 2 )
#define MW_ARG_MY_FLAGS Pt_RESOURCE( Pt_USER(1), 3 )
#define MW_ARG_MY_FLAG_BIT Pt_RESOURCE( Pt_USER(1), 4 )
#define MW_ARG_MY_POINT_ARRAY Pt_RESOURCE( Pt_USER(1), 5 )
#define MW_ARG_MY_TAG_DATA Pt_RESOURCE( Pt_USER(1), 6 )
#define MW_CB_MY_CALLBACK Pt_RESOURCE( Pt_USER(1), 7 )

#define MW_MY_FLAG_BIT 0x04000000

/* widget instance structure */
typedef struct my_widget{
    PtBasicWidget_t basic; //Subclass of PtBasic.
} ShadowedBoxWidget;
```
char character;
char *my_string;
short my_short;
long flags;
PhPoint_t *points;    //Array of points.
ushort_t num_points;
void *tag_data;
PtCallbackList_t *my_callbacks; //Linked list of callbacks.
} MyWidget_t;

In the class-creation function in the source code file we have:

```c
static const PtResourceRec_t resources = {
    Pt_ARG_POS, arg_pos_override, 0,
    Pt_ARG_IS_NUMBER( PtWidget_t, area.pos ), 0,
    MW_ARG_MY_CHARACTER, Pt_CHANGE_REDRAW, 0,
    Pt_ARG_IS_NUMBER( MyWidget_t, character ), 0,
    MW_ARG_MY_STRING, Pt_CHANGE_RESIZE_REDRAW, 0,
    Pt_ARG_IS_STRING( MyWidget_t, my_string ), 0,
    MW_ARG_MY_SHORT, set_my_short, get_my_short, 0, 0,
    MW_ARG_MY_FLAGS, Pt_CHANGE_INVISIBLE, 0,
    Pt_ARG_IS_FLAGS( MyWidget_t, flags ), 0,
    MW_ARG_MY_FLAG_BIT, Pt_CHANGE_INVISIBLE, 0,
    Pt_ARG_IS_BOOLEAN ( MyWidget_t, flags ),
    MW_MY_FLAG_BIT,
    MW_ARG_MY_POINT_ARRAY, Pt_CHANGE_RESIZE_REDRAW, 0,
    Pt_ARG_ISARRAY( MyWidget_t, points ),
    Pt_ARG_IS_NUMBER( MyWidget_t, num_points ),
    MW_ARG_MY_TAG_DATA, Pt_CHANGE_INVISIBLE, 0,
    Pt_ARG_IS_ALLOC( MyWidget_t, tag_data ), 0,
    MW_CB_MY_CALLBACK, Pt_CHANGE_INVISIBLE, 0,
    Pt_ARG_IS_CALLBACK_LIST( MyWidget_t, my_callback ), 0,
}
```

`Pt_ARG_POS` is inherited from `PtWidget`. Here we're overriding its `mod_f()` function with one of its own.

The `MW_ARG_MY_SHORT` resource and `mod_f()/query_f()` functions could look like this:

```c
static void set_my_short( PtWidget_t *widget, PtArg_t *argt )
{
    MyWidget_t *mw = (MyWidget_t *)widget;

    if( mw->my_short == (short)argt->value )
        return; // don’t do work if nothing is changing.
    mw->my_short = argt->value;

    // My widget needs to redraw when my_short changes...
    PtDamageWidget( widget );
}

static int get_my_short( PtWidget_t *widget, PtArg_t *argt )
```
{  
    MyWidget_t *mw = (MyWidget_t *)widget;  
    if( argt->value )  
    // The address of a pointer to a short is in argt->value  
        *(short **)argt->value = &mw->my_short;  
    else  
        argt->value = (long) mw->my_short;  
    return Pt_TRUE;  
    /* The only time one would return Pt_FALSE is if no data  
        is given as a result of the query */  
}

Defining the widget class

In Photon, a widget class is defined much a like a widget. You set up a list of  
arguments or class resources and call PtCreateWidgetClass(). This list is used to set  
the field members of the widget class structure that Photon uses to determine how to  
handle the widget.

Widget class structure

Let’s look at the general form of a PtWidgetClass_t structure:

```c
struct Pt_widget_class {  
    char *description;  
    struct Pt_widget_class *superclass;  
    PtWidgetClassRef_t *class_ref;  
    unsigned class_len;  
    unsigned state_len;  
    unsigned long flags;  
    void (*dflts_f)( PtWidget_t * );  
    int (*init_f)( PtWidget_t * );  
    int (*connect_f)( PtWidget_t * );  
    void (*unrealize_f)( PtWidget_t * );  
    void (*destroy_f)( PtWidget_t * );  
    void (*realized_f)( PtWidget_t * );  
    PtClassRawCallback_t const *callbacks;  
    int (*setres_f)( PtWidget_t *, int,  
                    PtArg_t const *,  
                    PtResourceRec_t const *rrec );  
    int (*getres_f)( PtWidget_t const *,  
                    int, PtArg_t * );  
    unsigned int ex_state_len;  
    struct Pt_resource_range *res_ranges;  
    void (*syncwidget_f)( PtWidget_t *widget );  
    int (*calc_region_f)( PtWidget_t *,  
                         unsigned int *,  
                         PhRegion_t *,  
                         PhRect_t * );  
    PtResourceRec_t const **res_all;  
    unsigned short res_nranges;  
    uchar_t num_actions;  
    unsigned char num_callbacks;  
    PtWidgetAction_t *actions;  
    short max_style_index;  
    unsigned short res_nall;  
    PtWidgetClassStyle_t **styles;  
    unsigned reserved[5];
};
```
Widget class structure description

Let’s look at the PtWidgetClass_t structure one member at time:

char *description

The name of the widget type, for example “PtTree” if the widget is a PtTree.

PtWidgetClass_t *superclass

A pointer to the superclass of this widget class. This value defines the superclass this class inherits from. If a widget class is at the top of the hierarchy, this pointer is NULL.

PtWidgetClassRef_t *class_ref

A pointer to the class_ref structure for this widget class.

unsigned class_len

The size of this widget class (this differs from the superclass only if the class extends the superclass definition).

unsigned state_len

The size of the widget instance structure for this widget class.

unsigned long flags

A general flag field for widget classes. For valid flag bits, see the list below.

void (*dflts_f)(PtWidget_t *)

Defines the Defaults method.

int (*init_f)(PtWidget_t)

Defines the Initialization method.

int (*connect_f)(PtWidget_t *)

Defines the Connection method.

void (*unrealized_f)(PtWidget_t *)

Defines the Unrealization method.

void (*destroy_f)(PtWidget_t *)

Defines the Destruction method.

void (*realized_f)(PtWidget_t *)

Defines the Realization method.

PtClassRawCallback_t *callbacks

The array of widget actions (raw callbacks) for the widget class. If a function in the callback list has an event mask matching the event being handled, that function is invoked.
int (*setres_f)(PtWidget_t *, int, PtArg_t *)

Defines the Set Resources method that’s called when a user calls PtSetResources() on a widget of this class.

int (*getres_f)(PtWidget_t *, int, PtArg_t *)

Defines the Get Resources method that’s called when a user calls PtGetResources() on a widget of this class.

unsigned int ex_state_len

The length of state_len. Used internally.

struct Pt_resource_range *res_ranges

Used internally.

void (*syncwidget_f)( PtWidget_t *widget )

Defines the Synchronize Widget method that’s called when a user calls PtSyncWidget() on a widget of this class.

int (*calc_region_f)( PtWidget_t *, unsigned int *, PhRegion_t *, PhRect_t * )

Defines the Calculate Region method that’s called when the Photon library calls PtCalcRegion() on a widget of this class.

PtResourceRec_t const **res_all

Defines the resource manifest table for the widget class. The resource manifest table is an array of pointers to all the resource records defined or inherited by this class.

unsigned short res_ranges

The length of res_ranges. Used internally.

uchar_t num_actions

The number of items in the actions array.

unsigned char num_callbacks

The number of items in the callbacks array.

PtWidgetAction_t *actions

A list of widget actions.

short max_style_index

The number of items in the styles array.

unsigned short res_nall

The number of items in the res_all array. Used internally.

PtWidgetClassStyle_t **styles;

An array of styles for this widget class.
unsigned reserved[ 5 ]

Reserved for future use.

These are the valid flag bits for the flags member:

Pt_CONTAINER The widget class is a container.

Pt_RECTANGULAR Rectangular widgets are opaque when filled. Opaque widgets don’t damage widgets below them when they’re modified (unless their size or position is modified).

Pt_FORCE_UNREALIZE
(set automatically when required)

The Unrealization method (unrealize_f()) for this class and its superclasses is called when this widget is unrealized.

Pt_DISJOINT
(c.e. PtWindow, PtMenu, PtRegion)

Indicates that widgets of this class own regions that aren’t children of the regions of their widget parents. This means that a disjoint widget is *not* assumed to sit inside its parent, and therefore is responsible for collecting exposes and emitting draw stream for itself and its non-disjoint descendants.

Pt_NO_INHERITED_RESOURCES

Prevents the search for a resource from walking up through superclasses. Only the resources for this class are handled; all others are ignored. This doesn’t prevent resources from being propagated to procreated widgets.

This is handy for allowing common resources such as Pt_ARG_COLOR to pass to procreated children without having to write a resource-redirector function.

Pt_OCCLUSIVE Drawing routines skip all children of a widget derived from an occlusive class. The rendering of these children is the responsibility of the occlusive widget.

Pt_BASIC Indicates that the widget is a PtBasic or subclass.

Pt_FORCE_SYNC Indicates widgets must be sized if the Pt_WIDGET_RESIZE flag is set during a set resource regardless of the widgets realized state.

**Widget class resource table**

Not all members of the widget class structure can be set directly; some are used internally by the widget library. The members you can change using the class resource manifests are:
For the description, you might include something like this in your class definition:

```c
static const PtArg_t args[] =
{
  ...
  { Pt_SET_DESCRIPTION, (long) "PtButton" },
  ...
};
```

In general, it’s assumed that the description corresponds to the widget’s name (i.e. applications might conceivably use this field to display the name of the widget’s class) and hence it’s recommended you follow this practise. If additional information is required, the convention adopted is to append a colon (:) followed by the text; applications that use this description field will look for the first : and parse based on it. So this would translate the above example to

```c
{ Pt_SET_DESCRIPTION, (long) "PtButton:dave was here" },
```
This is a convention, not a rule. You can use this field for whatever you want, as long as you realize that some applications might use this field (for informational purposes only). The most graceful way to have such applications disregard this field altogether is to precede the text with a a colon, like this:

```
{ Pt_SET_DESCRIPTION, (long) "dave was here" },
```
in which case, the application should indicate a “value not known” state (i.e. disregard anything past the : in all cases).

When defining a custom widget class, you can extend the definition of its superclass with new “class-level” methods and/or callbacks. For example, the PtWidget class is extended by PtBasic, which adds new class methods for providing focus notification. This was done as shown in the following excerpt from the PtBasic.h header file:

```c
typedef struct Pt_basic_widget_class {
    PtWidgetClass_t core;
    void (*got_focus_f)( PtWidget_t *,
                         PhEvent_t * );
    void (*lost_focus_f)( PtWidget_t *,
                         PhEvent_t * );
    void (*calc_opaque_f)( PtWidget_t *
                         );
} PtBasicWidgetClass_t;

#define Pt_SET_GOT_FOCUS_F (Pt_ARG_IS_POINTER(PtBasicWidgetClass_t,got_focus_f))
#define Pt_SET_LOST_FOCUS_F (Pt_ARG_IS_POINTER(PtBasicWidgetClass_t,lost_focus_f))
#define Pt_SET_CALC_OPAQUE_F (Pt_ARG_IS_POINTER(PtBasicWidgetClass_t,calc_opaque_f))
```

When a class definition is extended in this way, the new class structure size must be passed as the second parameter to PtCreateWidgetClass():

```
PtBasic->wclass = PtCreateWidgetClass( PtWidget,
                                        sizeof(PtBasicWidgetClass_t ),... );
```

If the class isn’t extended, the second parameter may be passed as 0. Also, when the class is extended, you must define new Pt_SET* manifests to provide access to the new class members from the create class function.

**PtBasic class resource additions**

The PtBasic widget class provides the following three additional structure members:

<table>
<thead>
<tr>
<th>Member</th>
<th>Description</th>
<th>Resource manifest</th>
</tr>
</thead>
<tbody>
<tr>
<td>got_focus_f</td>
<td>Got Focus method</td>
<td>Pt_SET_GOT_FOCUS_F</td>
</tr>
<tr>
<td>lost_focus_f</td>
<td>Lost Focus method</td>
<td>Pt_SET_LOST_FOCUS_F</td>
</tr>
<tr>
<td>calc_opaque_f</td>
<td>Calc Opaque Rect method</td>
<td>Pt_SET_CALC_OPAQUE_F</td>
</tr>
</tbody>
</table>
Let’s look at the class resource table from our ShadowedBox example in more detail:

```c
static PtArg_t args[] = {
    { Pt_SET_VERSION, 110 },
    { Pt_SET_STATE_LEN, sizeof( ShadowedBoxWidget ) },
    { Pt_SET_DFLTS_F, (long)shadowedbox_dflts },
    { Pt_SET_DRAW_F, (long)shadowedbox_draw },
    { Pt_SET_FLAGS, 0, Pt_RECTANGULAR },
    { Pt_SET_NUM_RESOURCES,
      sizeof( resources ) / sizeof( resources[0] ) },
    { Pt_SET_RESOURCES, (long)resources,
      sizeof( resources ) / sizeof( resources[0] ) },
};
```

Pt_SET_VERSION Tells the widget library what style this widget is. For your widgets, this value is always at least 110 (110 corresponds to Photon 1.10 style).

Pt_SET_STATE_LEN Defines the length of the widget instance structure.

Pt_SET_DFLTS_F Defines the Defaults method, which is called to set default values for the widget when it’s first created. In our example, this function is called `shadowedbox_dflts()`.

Pt_SET_DRAW_F Defines the Draw method (e.g. `shadowedbox_draw()`), which is called when the widget needs to be drawn.

Pt_SET_FLAGS Defines behavioral flags related to this widget class. This example turns off the Pt_RECTANGULAR flag, which indicates the widget can’t take advantage of flicker-free, opaque-drawing methods.

Pt_SET_NUM_RESOURCES Defines the number of resources in the table.

Pt_SET_RESOURCES Defines the table of resources unique to this widget, as described above.

## Class methods

This section describes the role and responsibilities of the class methods defined in the widget class structure. The fundamental methods defined by `PtWidget` are:

- Defaults
- Initialization
- Extent
- Connection
• Realization
• Draw
• Unrealization
• Destruction
• Set Resources
• Get Resources

The PtBasic widget extends these with three additional methods:
• Got Focus
• Lost Focus
• Calc Opaque Rect

These are the methods you write to define the functionality of your widget. Not all methods need to be defined and coded by a widget class; they can be inherited from a superclass. Almost all widgets need at least the Draw method, since the primary purpose of building a custom widget is to draw something.

Defaults method

Type: Chained down, unstoppable

This method sets the default values for all widget instance structure members. It’s called during the creation of a widget instance (call to PtCreateWidget()). When a widget instance is first created, all members of the instance structure are zeroed out and then the Defaults method for each parent class (starting from PtWidget at the top) are called in sequence from top to bottom. This allows the lower-level classes to override the default values set by the parent classes in the hierarchy. Here’s an example showing how to initialize PtBasic variables:

```c
static void basic_dflts( PtWidget_t *widget )
{
    PtBasicWidget_t *basic = (PtBasicWidget_t *) widget;
    widget->border_width = 2;
    widget->cursor_color = Ph_CURSOR_DEFAULT_COLOR;
    widget->resize_flags |= Pt_RESIZE_XY_AS_REQUIRED;

    basic->color = Pg_BLACK;
    basic->fill_color = Pg_GREY;
    basic->top_border_color = Pg_WHITE;
    basic->bot_border_color = Pg_DGREY;
    basic->flags = Pt_DAMAGE_ON_FOCUS;
}
```

Although the instance variables in this example are specific to PtBasic, the initialization technique is common to all Defaults methods.
Initialization method

Type: Chained up, stoppable

This is the first method called when a widget is realizing (call to PtRealizeWidget()). The Initialization method does final checks to ensure the widget is “extentable.” This check ensures all members used in the subsequent Extent method are correctly assigned.

The Initialization method is also the best place to create unexported subordinate widgets. Unexported subordinate widgets are used by the widget but not available to the user. For example, a widget such as PtPrintSel uses other widgets (PtText, PtLabel, and PtButton) but doesn’t allow the user to act directly on the subordinate widgets. Instead, the widget provides new unique resources used internally to set the subordinate widgets. For more information about exported subordinate widgets, see “Compound widget anatomy” later in this chapter.

Initialization is executed each time the widget is realized. The Initialization method is called first for this widget followed by its parent superclass and so on until the core widget class (PtWidget) is reached. Any widget class in the chain can terminate the chaining process by returning Pt_END.

Here’s an example showing how a PtContainer widget is initialized:

```c
static in PtContainerInit( PtWidget_t *widget )
{
    PtContainerRegister( widget );
    return( Pt_CONTINUE );
}
```

The Initialization method is also used to register widgets for balloon handling. All container widgets provide a default balloon-handling mechanism. If you want your widget to support popup help balloons, you must register the widget with the container in this function. Here’s an excerpt from the PtLabel code:

```c
static int label_init( PtWidget_t *widget )
{
    PtLabelWidget_t *label = (PtLabelWidget_t *) widget;
    PtBalloonCallback_t bcalls;
    PtArg_t argt;

    if( (label->flags & Pt_SHOW_BALLOON)
        && ( !( label->flags & Pt_BALLOON_REGISTERED ) ) )
    {
        bcalls.widget = widget;
        bcalls.event_f = label_balloon_callback;
        PtSetArg( &argt, Pt_CB_BALLOONS, &bcalls, 0 );
        PtSetResources( widget->parent, 1, &argt );
        label->flags |= Pt_BALLOON_REGISTERED;
    }
    return Pt_CONTINUE;
}
```
Extent method

Type: Inherited

This is the second method called during the realization of a widget (call to
PtRealizeWidget()). The Extent method is responsible for determining the widget’s
screen real estate (bounding box) with regard to its resize policy. The Extent method is
called frequently during the life of a widget — whenever the widget is moved or
resized, or resources marked as Pt_CHANGE_RESIZE or
Pt_CHANGE_RESIZE_REDRAW are modified.

The following are examples of these types of resources, as defined by the PtWidget
class:

```c
static PtResourceRec_t resources[] = {
{ Pt_ARG_AREA, Pt_CHANGE_RESIZE, 0,
  Pt_ARG_IS_STRUCT(PtWidget_t, area) },
{ Pt_ARG_POS, Pt_CHANGE_RESIZE, 0,
  Pt_ARG_IS_STRUCT(PtWidget_t, area.pos) },
{ Pt_ARG_DIM, Pt_CHANGE_RESIZE, 0,
  Pt_ARG_IS_STRUCT(PtWidget_t, area.size) },
};
```

In addition, the Extent method is called whenever any other resource causes a widget
to change size or position. The following example code is taken from PtLabel:

```c
static PtResourceRec_t resources[] = {
{ Pt_ARG_LABEL_TYPE, Pt_CHANGE_RESIZE_REDRAW, 0,
  Pt_ARG_IS_NUMBER(PtLabelWidget_t, type), 0 },
};
```

The widget engine uses the extent calculated in this method to determine when this
widget is involved in an event, when the widget needs to be repaired, and which other
widgets need to be repaired when this one changes.

Every widget class inherits the Extent method of its superclass. If this method is
suitable, you won’t need to provide your own Extent method. If the superclass’s
Extent method isn’t quite what your widget needs, you should provide only what the
widget’s superclass doesn’t provide, then call the Extent method of the superclass to
do the rest of the work.

It’s unusual to write an Extent method that calculates the entire extent without using
the superclass’s method. Quite often you’ll want most, but not all, of the superclass’s
extent behavior. There’s usually a way to prevent the Extent method of the superclass
from performing any set of extenting behaviors (see the “Extent method” sections in
the Using Widget Superclasses chapter for details).

Extenting a widget usually involves four stages:

1. Calculate the canvas. Check the chapter on Using Widget Superclasses for
canvas services. The canvas is the area within which a widget must restrict its
rendering. The canvas function is PtCalcCanvas().
2 Calculate the render rectangle. This is the bounding box area required for this widget to display all its data, including points, widget children, any text strings, etc.

3 Call \texttt{PtResizeCanvas()} passing to it the widget and desired size. 
\texttt{PtResizeCanvas()} applies the widget’s resize policy automatically and modifies the widget’s dimensions as required to ensure the widget conforms to its resize policy.

If a widget’s canvas can’t be resized to encompass the render rectangle, \texttt{PtResizeCanvas()} sets the Pt\_UCLIP bit in the widget’s resize flags. The widget’s Draw method should check the Pt\_UCLIP bit. If this bit is set, the Draw method should apply clipping to prevent the widget from rendering outside its canvas.

4 Call:
\texttt{PtSuperClassExtent( PtBasic, widget )}
to have the final extent calculation performed. The Extent method of \texttt{PtBasic} uses the widget’s position, dimension, border width, and flags to determine the extent.

Here’s an example showing how to apply these guidelines:

```c
mywidget_extent( PtWidget_t *widget )
{
    MyWidgetUnion_t *mwu = (MyWidgetUnion_t *)widget;
    PhRect_t canvas, render;
    PhDim_t size;

    PtCalcCanvas( widget, &canvas );
    render.ul = render.lr = canvas.ul;
    PgExtentText( &render, &render.ul, mwu->label.font,
        mwu->label.string, 0 );
    size.x = render.lr.x - render.ul.x + 1;
    size.y = render.lr.y - render.ul.y + 1;
    PtResizeCanvas( widget, &size );
    PtSuperClassExtent( PtBasic, widget );
}
```

### Connection method

Type: Chained up, stoppable

This method creates Photon regions whenever a widget requires them. Generally, it isn’t necessary to explicitly create your own region since the Connection method of \texttt{PtWidget} does this for you.

However, it may be beneficial to create your own region if you need to modify the region based on the current environment. For example, \texttt{PtMenu} widgets have to modify a region whenever they’re displayed:

```c
static int menu_connect( PtWidget_t *widget )
{
    PhRegion_t region;
```
unsigned   fields;
PhRect_t    rect;
PtMenuWidget_t *menu = (PtMenuWidget_t *)widget;
PtWidget_t   *wp;

  /* calculate menu region */
  fields = UINT_MAX;
  if( !PtCalcRegion( &fields, widget, &region, &rect ) )
      return( -1 );

  /* open menu region */
  region.parent   = menu->ff_wgt->rid;
  region.events_opaque |= Ph_EV_DRAW | Ph_EV_PTR_ALL;
  region.events_opaque &= ~Ph_EV_KEY;
  region.events_sense |= Ph_EV_PTR_ALL;
  region.events_sense &= ~Ph_EV_KEY;
  region.flags = 0;
  fields |= Ph_REGION_PARENT | Ph_REGION_EV_SENSE |
             Ph_REGION_EV_OPAQUE;
  fields &= ~( Ph_REGION_BEHIND | Ph_REGION_IN_FRONT );
  widget->rid = PhRegionOpen( fields, &region, &rect, NULL );
  wp = widget;
  while( PtWidgetIsClass( wp, PtMenu ) )
  {
    menu_pdr_start( wp );
    if( (!wp->parent )
    {
      break;
      wp = wp->parent->parent;
    }
    if( widget->parent &&
        PtWidgetIsClassMember( widget->parent, PtContainer ) )
    {
      if( !((menu->flags & Pt_MENU_CHILD) )
      {
        // If the menu is off a window, focus that window.
        PhEvent_t event;
        memset( &event, 0, sizeof( event ) );
        menu->prev_focus =
            PtContainerFindFocus( widget->parent );
        if( widget->parent->class_rec->flags & Pt_DISJOINT )
            PtContainerNullFocus( widget->parent, &event );
        else
        {
          int flags = widget->parent->flags;
          widget->parent->flags |= Pt_GETS_FOCUS;
          PtContainerGiveFocus( widget->parent, &event );
          widget->parent->flags = flags;
        }
      } else
      {
        menu->flags |= Pt_MENU_SUBFOCUS;
        ((PtContainerWidget_t *)widget->parent->focus = widget;
      }
      wp = PtFindDisjoint( widget );
      for( wp = wp->parent;
          wp && (PtWidgetIsClassMember(wp, PtMenu)
            || PtWidgetIsClassMember(wp, PtMenuButton))
          wp = wp->parent);
    if( wp )
    {
      wp = PtFindDisjoint( wp );
      if( PtWidgetIsClassMember( wp, PtWindow )
          (PtContainerWidget_t *)wp)->last_focus = NULL;
      if( !(PtWindowGetState( wp ) & Ph_WM_STATE_ISFOCUS ) )
      {
        PtWindowFocus( wp );
      }  
    }
  }
If you create your own region, the Connection method of PtWidget won’t create another region — it modifies the current region. To prevent modification of the current region, have the widget class return Pt_END.

Realization method

Type: Inherited

This method is called after a widget has been realized. Its function is similar to that of the Pt_CB_REALIZED callback, but unlike the Pt_CB_REALIZED callback, it can’t be accessed by any developer using your widget.

This method is used primarily by compound widgets to perform any postrealize operations, such as realizing any subordinate widgets that couldn’t be realized to this point. It’s the last method to be called prior to the Draw method, and it’s invoked just prior to the user’s Pt_CB_REALIZED callbacks. For more information, see the section on “Compound widget anatomy.”

Draw method

Type: Inherited

This is the last method called during the realization process. The Draw method is used to render the widget on the screen during realization and is called to redraw a widget afterwards whenever that widget is damaged.

When a widget is damaged, a pointer to the damaged widget and a list of tiles describing the damage are passed to the Draw method. The first tile in the damage list reveals the total extent of the damage (its bounding box encompasses all remaining tiles). The remaining tiles contain the actual extents damaged. Unless the widget is complex or passes a lot of draw data (large images) and can be sped up by drawing damaged areas only, you should ignore the damage list.

The widget library uses the damage list to clip parts that shouldn’t be drawn. If you plan to use the damage tiles, make sure to translate the widget canvas using the widget’s offset. Use PtWidgetItemOffset() to obtain the offset — the damage list is relative to the disjoint parent widget (usually a PtWindow widget).

The Draw method restricts its updates to the canvas of the damaged widget. This is done by setting a clipping rectangle if necessary. Here’s a drawing example taken from the code for PtButton:

```c
static void button_draw( PtWidget_t *widget, PhTile_t *damage )
{
    PtButtonWidget_t *button = (PtButtonWidget_t *)widget;
    PgColor_t tcolor;
```
button->label.data = button->unarmed_data;
/* set fill_color to arm_color if required */
if ( widget->flags & Pt_SET )
{
    if( button->arm_fill == 1 )
    {
        tcolor = button->label.basic.fill_color;
        button->label.basic.fill_color = button->arm_color;
    }
    if( button->arm_data )
        button->label.data = button->arm_data;
}

/* draw button - includes highlight */
PtSuperClassDraw( PtLabel, widget, damage );

/* restore fill_color */
if ( widget->flags & Pt_SET )
{
    if ( button->arm_fill == 1 )
        button->label.basic.fill_color = tcolor;
}
}

The following code excerpt (from PtArc) shows how setting the Pt_UCLIP bit in the widget’s resize flags affects clipping:

PtCalcCanvas( widget, &rect );
if ( widget->resize_flags & Pt_UCLIP )
    PtClipAdd( widget, &rect );
PgDrawArc( &pos, &dim, start, end, arc->type | flags );
if ( widget->resize_flags & Pt_UCLIP )
    PtClipRemove( );

Using the Pg library safely

You’ll use the Pg* functions in your widget’s Draw method, but you need to use them safely. Here are some things to remember:

PgSetClipping()
PgSetMultiClip()
PgSetUserClip() These are used by the library. The only safe way of changing clipping in a widget’s Draw method is to use PtClipAdd() and PtClipRemove() (see the Widget Building Library API chapter).

PgSetDrawMode()
PgSetPlaneMask()
PgSetStrokeCap()
PgSetTranslation()
PgSetUnderline() These aren’t set by the library, but widgets assume they’re set to their default values. If your Draw method changes any of them, you have to restore it.
Class methods

These are automatically reset to the default after a widget’s Draw method returns. It’s safe to change them.

These are automatically reset to the default after a widget’s Draw method returns. It’s safe to change them.

These are automatically reset to the default after a widget’s Draw method returns. It’s safe to change them.

These are automatically reset to the default after a widget’s Draw method returns. It’s safe to change them.

Unrealization method

Type: Chained up

This method is called when a widget is being unrealized. The Unrealization method is responsible for removing any regions created by a widget (widget->rid is removed automatically by the PtWidget class’s Unrealization method). The Unrealization method should free any memory that would be reallocated during the realization process.

The Unrealization method is also used to deregister widgets. For example, if a label widget registered a balloon in its Initialization method, it must deregister the balloon in the Unrealization method or the balloon will try to inflate whenever the mouse pointer pauses over the last location of the widget. Here’s an example taken from the code for PtLabel:

```c
static int label_unrealize(PtWidget_t *widget )
{ PtLabelWidget_t *label = (PtLabelWidget_t*)widget;
  PtBalloonCallback_t bcalls;
  PtArg_t arg;

  if(label->balloon_widget)
    PtDestroyWidget( label->balloon_widget );

  bcalls.widget = widget;
  bcalls.event_f = label_balloon_callback;
```
if( label->flags & Pt_SHOW_BALLOON ){
    PtSetArg( &arg, Pt_CB_BALLOONS, &bcalls, Pt_LINK_DELETE );
    PtSetResources( widget->parent, 1, &arg );
}
label->flags &= ~Pt_BALLOON_REGISTERED;
return Pt_CONTINUE;

Destruction method

Type: Chained up, unstoppable

This method is called when a widget is being destroyed by the application. The Destruction method is responsible for releasing all resources allocated by a widget class during its lifetime. The Destruction method doesn’t deal with memory allocated by a widget’s superclass, because each class is responsible for freeing its own memory. Here’s an example taken from the code for PtLabel:

static int
label_destroy( PtWidget_t *widget )
{
    PtLabelWidget_t *label = (PtLabelWidget_t *)widget;
    PtArg_t arg;
    PtBalloonCallback_t bcalls;
    if( label->flags & Pt_BALLOON_REGISTERED ) {
        bcalls.widget = widget;
        bcalls.event_f = (void*)label_balloon_callback;
        if( label->flags & Pt_SHOW_BALLOON )
            { 
                PtSetArg( &arg, Pt_CB_BALLOONS, &bcalls, Pt_LINK_DELETE );
                PtSetResources( widget->parent, 1, &arg );
            }
        label->flags &= ~Pt_BALLOON_REGISTERED;
    }
    return 0;
}

Set Resources method

Type: Inherited

This method is used to take over the standard resource setting process built into the Photon library. Compound widgets are the only widgets that set this method (see the section on “Compound widget anatomy”).

Get Resources method

Type: Inherited

This method is used to take over the standard resource retrieval process built into the Photon library. Compound widgets are the only widgets that set this method (see the section on “Compound widget anatomy”).
Got Focus method

Type: Inherited

This method is used by subclasses of PtBasic only. It’s called when a widget gets focus. If your widget gets focus in such a way that the whole widget need not be redrawn, clear the Pt_DAMAGE_ON_FOCUS flag (from PtBasic) and damage the appropriate area when your widget gets focus.

This method isn’t chained. If your class defines a Got Focus method, you’ll have to either call PtSuperClassLostFocus() to preserve automatic highlighting and widget-level Got Focus method behavior or implement the behavior in the Got Focus method of your class. Here’s an excerpt from the code for PtBasic:

```c
static basic_got_focus( PtWidget_t *widget, PhEvent_t *event)
{
    PtBasicWidget_t *basic = (PtBasicWidget_t *) widget;
    PtCallbackInfo_t cbinfo;
    PtArg_t arg;

    /* damage the widget so that focus rendering
       will take effect */
    if( (widget->flags & Pt_FOCUS_RENDER) &&
        (basic->flags & Pt_DAMAGE_ON_FOCUS))
        PtDamageWidget( widget);

    /* setup callback structure */
    cbinfo.reason_subtype = 0;
    cbinfo.event = event;
    cbinfo.cbdata = NULL;

    /* an autohighlight widget with focus should invoke ARM */
    if( widget->flags & Pt_AUTOHIGHLIGHT )
    {
        PtSetArg( &arg, Pt_ARG_FLAGS,
                   Pt_HIGHLIGHTED, Pt_HIGHLIGHTED);
        PtSetResources( widget, 1, &arg);
        cbinfo.reason = Pt_CB_ARM;
        PtInvokeCallbackList( basic->arm, widget, &cbinfo);
    }

    /* invoke got focus callback */
    cbinfo.reason = Pt_CB_GOT_FOCUS;
    PtInvokeCallbackList( basic->got_focus, widget, &cbinfo);
    return Pt_CONTINUE;
}
```

Lost Focus method

Type: Inherited

This method is used by subclasses of PtBasic only. It’s called when a widget loses focus. This method isn’t chained. If your class defines a Lost Focus method, you’ll have to either call PtSuperClassLostFocus() to preserve automatic highlighting and widget-level Lost Focus method behavior or implement the behavior in the Lost Focus method of your class. Here’s an excerpt from the code for PtBasic:
static basic_lost_focus( PtWidget_t *widget, PhEvent_t *event )
{
  PtBasicWidget_t *basic = ( PtBasicWidget_t *) widget;
  PtCallbackInfo_t cbinfo;
  PtArg_t arg;
  PhRect_t wrect, rect = widget->extent;

  if( (widget->flags & Pt_FOCUS_RENDER) &&
      ( basic->flags & Pt_DAMAGE_ON_FOCUS ) )
    if( basic->fill_color == Pg_TRANSPARENT )
      {
        PtTranslateRect( &rect,
            (PhPoint_t*)PtCalcCanvas( widget->parent,
            &wrect ) );
        PtDamageExtent( widget->parent, &rect );
      } else
        PtDamageWidget( widget );

  cbinfo.reason_subtype = 0;
  cbinfo.event = event;
  cbinfo.cbdata = NULL;
  if( widget->flags & Pt_AUTOHIGHLIGHT )
    {
      PtSetArg( &arg, Pt_ARG_FLAGS, 0, Pt_HIGHLIGHTED );
      PtSetResources( widget, 1, &arg );
      cbinfo.reason = Pt_CB_DISARM;
      PtInvokeCallbackList( basic->disarm, widget, &cbinfo );
    }
  cbinfo.reason = Pt_CB_LOST_FOCUS;
  PtInvokeCallbackList( basic->lost_focus, widget, &cbinfo );
  return Pt_CONTINUE;
}

Calc Opaque Rect method

Type: Inherited

This method is used only by subclasses of PtBasic. It sets or clears the widget’s Pt_OPAQUE flag (Pt_ARG_FLAGS resource) and the Pt_RECTANGULAR widget class flag.

When the Pt_OPAQUE flag is set for a widget, it means the widget draws over the entire widget extent area. This allows the widget library to be smart about redrawing the widget, because it knows that nothing beneath the widget needs to be redrawn. This flag is essential for creating flicker-free effects. If any part of the widget is transparent (i.e. any widget beneath can be seen), the Pt_OPAQUE flag must be cleared. Here’s an excerpt from the code for PtBasic:

static void basic_calc_opaque( PtWidget_t *widget )
{
  PtBasicWidget_t *basic = ( PtBasicWidget_t *) widget;

  /* if widget is transparent or round it can’t be opaque */
  if ( basic->fill_color == Pg_TRANSPARENT | |
     basic>roundness )
    widget->flags &= ~Pt_OPAQUE;
  else
    /* must have RECTANGULAR class flag set */
    if ( widget->class_rec->flags &
        Pt_RECTANGULAR )
    {
```
widget->flags |= Pt_OPAQUE;
basic_opaque_rect( widget );
}
}

static void basic_opaque_rect( PtWidget_t *widget )
{
if( widget->flags & Pt_HIGHLIGHTED )
    memcpy( &widget->opaque_rect,
        &widget->extent, sizeof( PhRect_t ) );
else
    PtCalcCanvas( widget, &widget->opaque_rect );
}

Widget actions

Widget actions are used to make a widget interactive. For example, when you click a
button widget, you can see it press in and then pop out again. This is achieved by
setting up raw callbacks sensitive to specific Photon events. The way a widget
interacts with events defines the widget’s behavior.

The callbacks list, Pt_SET_RAW_CALLBACKS (see the “Widget class resource table”
section earlier in this chapter), defines a widget’s response if it’s different from the
behavior of its superclasses. This method is defined in the same manner as
Pt_CB_RAW. The primary difference is that the class’s raw callback list is invoked
before the user’s raw callback list. Here’s an excerpt from the code for PtBasic:

```
static const PtClassRawCallback_t callback = {
    Ph_EV_BUT_PRESS | Ph_EV_BUT_RELEASE |
    Ph_EV_BUT_REPEAT | Ph_EV_BOUNDARY,
    basic_callback
};

static PtArg_t args[] = {
    ...
    { Pt_SET_RAW_CALLBACKS, &callback },
    ...
};
```

In the example above, whenever the widget receives one of the four events
(Ph_EV_BUT_PRESS, Ph_EV_BUT_RELEASE, Ph_EV_BUT_REPEAT, or
Ph_EV_BOUNDARY), the basic_callback() function is invoked. This function can
check the event type and act accordingly.

For clarity, let’s walk through a simplified description of the “click” process of a
PtButton widget.

First, PtButton receives the Ph_EV_BUT_PRESS event. The widget interprets the
press event, changes the widget flags to Pt_SET (possibly causing the widget to be
damaged and redrawn), and then invokes the Pt_CB_ARM callback.

When the user releases the mouse button, PtButton receives a
Ph_EV_BUT_RELEASE release event. The widget clears the Pt_SET flag and then
invokes the Pt_CB_DISARM and Pt_CB_ACTIVATE callbacks.
The action `PtButton` takes when the button is released is a little more complicated, because the widget actually doing the work is `PtBasic`, not `PtButton`. The `PtButton` class doesn’t define any raw callbacks because the handling done by `PtBasic` is sufficient. The `PtBasic` class handles the common Photon arm, disarm, repeat, activate, and menu callbacks for all widgets.

**Raw callback list**

Type: Chained up, stoppable

The class’s list of raw callbacks is used to make a widget sensitive to raw Photon event messages. This allows the widget to define a specific behavior related to external events or user interaction.

Returning `Pt_HALT` from a class’s raw callback prevents any superclass (or the user) from processing the event. The event is propagated up through the widget hierarchy and may be handled by a parent widget.

Returning `Pt_END` from a class’s raw callback prevents any superclass, user, or parent widget from processing the event. This is called consuming the event. Here’s an example taken from the code for `PtTimer`:

```c
static int timer_callback( PtWidget_t *widget, PhEvent_t *event )
{
    PtTimerWidget_t *timer = (PtTimerWidget_t *)widget;
    {
        PtCallbackInfo_t cbinfo;
        cbinfo.event = event;
        cbinfo.cbdata = NULL;
        cbinfo.reason = Pt_CB_TIMER_ACTIVATE;
        if( timer->state == Pt_TIMER_INITIAL )
            cbinfo.reason_subtype = Pt_TIMER_INITIAL;
        else
            cbinfo.reason_subtype = Pt_TIMER_REPEAT;
        timer->state = Pt_TIMER_REPEAT;
        PtInvokeCallbackType( widget, Pt_CB_TIMER_ACTIVATE, &cbinfo );
        if(timer->msec_repeat && timer->state == Pt_TIMER_REPEAT)
            PtTimerArm(widget,timer->msec_repeat);
    }
    return( Pt_CONTINUE );
}
```

.Retrofitting class-creation function

```c
static PtWidgetClass_t *PtCreateTimerClass( void )
{
    static const PtResourceRec_t resources[] =
    {
        { Pt_ARG_TIMER_INITIAL, timer_modify, 0, Pt_ARG_IS_NUMBER( PtTimerWidget_t, msec_value ) },
        { Pt_ARG_Timer_REPEAT, timer_modify, 0, Pt_ARG_IS_NUMBER( PtTimerWidget_t, msec_repeat ) },
        { Pt_ARG_AREA, Pt_CHANGE_PREVENT, 0, Pt_ARG_IS_STRUCT( PtTimerUnion_t, core.area ) },
        { Pt_ARG_DIM, Pt_CHANGE_PREVENT, 0, Pt_ARG_IS_STRUCT( PtTimerUnion_t, core.area.size ) },
        { Pt_ARG_POS, Pt_CHANGE_PREVENT, 0, Pt_ARG_IS_STRUCT( PtTimerUnion_t, core.area.pos ) },
        { Pt_CB_TIMER_ACTIVATE, Pt_CHANGE_INVISIBLE, 0, Pt_ARG_IS_IN_CB_LISTS( PtCallback_t ) },
    };
    static const PtClassRawCallback_t callback =
    {
        Ph_EV_TIMER, timer_callback
    };
    return( PtCreateWidgetClass( &resource, &callback ) );
}
```

//
Container widget anatomy

This section applies to creating PtContainer class widgets:

\[ PtWidget \rightarrow PtBasic \rightarrow PtContainer \rightarrow MyContainer \]

Child-constraint support

Container widgets extend the PtBasic widget class definition with a number of constraint methods. These methods allow the container to adjust itself automatically according to changes made to its children.

The container class extension is shown in the example code below (taken from the PtContainer.h header file):

```c
typedef struct Pt_container_widget_class {
    PtBasicWidgetClass_t basic;
    void (*child_created_f)( ... );
    int (*child_settingresource_f)( ... );
    int (*child_gettingresource_f)( ... );
    int (*child_realizing_f)( ... );
    void (*child_realized_f)( ... );
    void (*child_unrealizing_f)( ... );
    void (*child_unrealized_f)( ... );
    void (*child_destroyed_f)( ... );
    void (*child_move_resize_f)( ... );
    int (*child_getting_focus_f)( ... );
    int (*child_losing_focus_f)( ... );
    PtWidget_t * (*child_redirect_f)( ... );
} PtContainerClass_t;
```

Constraint methods can be individually enabled or disabled by setting or clearing the appropriate container->flags bits. You can enable or disable all the constraint methods at once by setting or clearing Pt_IGNORE_CONSTRAINTS.
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>Resource manifest</th>
</tr>
</thead>
<tbody>
<tr>
<td>child_created_f</td>
<td>Child Created</td>
<td>Pt_SET_CHILD.CREATED_F</td>
</tr>
<tr>
<td>child_settingresource_f</td>
<td>Child Setting Resource</td>
<td>Pt_SET_CHILD_SETTINGRESOURCE_F</td>
</tr>
<tr>
<td>child_gettingresource_f</td>
<td>Child Getting Resource</td>
<td>Pt_SET_CHILD_GETTINGRESOURCE_F</td>
</tr>
<tr>
<td>child_realizing_f</td>
<td>Child Realizing</td>
<td>Pt_SET_CHILD_REALIZING_F</td>
</tr>
<tr>
<td>child_realized_f</td>
<td>Child Realized</td>
<td>Pt_SET_CHILD_REALIZED_F</td>
</tr>
<tr>
<td>child_unrealizing_f</td>
<td>Child Unrealizing</td>
<td>Pt_SET_CHILD_UNREALIZING_F</td>
</tr>
<tr>
<td>child_unrealized_f</td>
<td>Child Unrealized</td>
<td>Pt_SET_CHILD_UNREALIZED_F</td>
</tr>
<tr>
<td>child_destroyed_f</td>
<td>Child Destroyed</td>
<td>Pt_SET_CHILD_DESTROYED_F</td>
</tr>
<tr>
<td>child_move_resize_f</td>
<td>Child Moved/Resized</td>
<td>Pt_SET_CHILD_MOVED_RESIZED_F</td>
</tr>
<tr>
<td>child_getting_focus_f</td>
<td>Child Getting Focus</td>
<td>Pt_SET_CHILD_GETTING_FOCUS_F</td>
</tr>
<tr>
<td>child_losing_focus_f</td>
<td>Child Losing Focus</td>
<td>Pt_SET_CHILD_LOSING_FOCUS_F</td>
</tr>
<tr>
<td>child_redirect_f</td>
<td>Child Redirection</td>
<td>Pt_SET_CHILD_REDIRECT_F</td>
</tr>
</tbody>
</table>

For convenience, the Photon widget building library lets you call these methods from a superclass using the provided functions. For more information, see the Widget Building Library API chapter.

**Child Created method**

Type: Inherited  
Constraint flag: Pt_CHILD_CREATED

Called whenever a new child is being created in this widget or one of its subordinate widgets. Here’s an excerpt taken from the code for PtGroup:

```c
static void child_created( PtWidget_t *widget,
                           PtWidget_t *child )
{
    PtCallback_t callback = { group_exclusive_callback,
                               NULL };
    PtGroupUnion_t *group = (PtGroupUnion_t *)widget;
    PtArg_t argt[2];
    int n = 0;

    callback.data = widget;

    if( group->basic.activate ) {
        PtSetArg( &argt[n++], Pt_CB_ACTIVATE,
                   &group->basic.activate->cb, 0 );
    }
    if( group->group.group_flags & Pt_GROUP_EXCLUSIVE ) {
        PtSetArg( &argt[n++], Pt_CB_ACTIVATE, &callback, 0 );
    }
    if( n )
        PtSetResources( child, n, argt );
}
```
Container widget anatomy

© 2007, QNX Software Systems GmbH & Co. KG.

**Child Realizing method**

Type: Inherited
Constraint flag: Pt_CHILD_REALIZING

Called whenever a child is in the process of being realized below this container in the hierarchy.

**Child Realized method**

Type: Inherited
Constraint flag: Pt_CHILD_REALIZED

Called whenever a child is realized below this container in the hierarchy.

**Child Moved/Resized method**

Type: Inherited
Constraint flag: Pt_CHILD_MOVED_RESIZED

Called whenever a child is moved or resized below this container in the hierarchy.

**Child Unrealizing method**

Type: Inherited
Constraint flag: Pt_CHILD_UNREALIZING

Called whenever a child is in the process of being unrealized below this container in the hierarchy.

**Child Unrealized method**

Type: Inherited
Constraint flag: Pt_CHILD_UNREALIZED

Called whenever a child is unrealized below this container in the hierarchy.

**Child Destroyed method**

Type: Inherited
Constraint flag: Pt_CHILD_DESTROYED

Called whenever a direct child of this widget or a direct child of one of its subordinates is destroyed.

**Child Setting Resource method**

Type: Inherited
Constraint flag: Pt_CHILD_SETTING_RESOURCE

Called whenever a resource is being set on a direct child of this widget or one of its subordinates.
Child Getting Resource method

Type: Inherited
Constraint flag: Pt_CHILD_GETTING_RESOURCE

Called whenever a resource is being retrieved from a direct child of this widget or one of its subordinates.

Child Getting Focus method

Type: Inherited
Constraint flag: Pt_CHILD_GETTING_FOCUS

Called whenever a child of this widget or one of its subordinates is about to be given focus.

Child Losing Focus method

Type: Inherited
Constraint flag: Pt_CHILD_LOSING_FOCUS

Called whenever a child of this widget or one of its subordinates is about to lose focus.

Child Redirection method

Type: Inherited
Constraint flag: Pt_CHILD_REDIRECTOR

Container widgets provide a mechanism for redirecting child widgets as they’re added to a container. You can use this mechanism to prevent certain widget classes from being added as direct children or redirect those children to other containers within the container (this feature is typically used by PtCompound widgets).

The child-redirector function is specified in the class-creation function and indicates where children should be attached in the widget hierarchy. Redirection is achieved by setting the child_redirect_f member of the Pt_container_widget_class structure to the child-redirector function.

The arguments to the child-redirector function are a widget pointer and the class type of the widget being added to the container. Using this information, the child-redirector function determines whether to accept the widget into the container or redirect the widget to another container, such as the parent of the container widget.

A good example is the PtMenuBar widget. This widget accepts only PtMenuButton widgets. Its class-creation function includes this:

```c
{ Pt_SET_CHILD_REDIRECT_F, (long)menubar_redirect },
```

The menubar_redirect() function would look like this:

```c
static PtWidget_t *menubar_redirect(
    PtWidget_t *menubar,
    PtWidgetClassRef_t *cref )
{
    if (cref != PtMenuButton )
        return menubar->parent;
```
The widget returned by the function becomes the parent for the widget being added.

You can use `PtCompoundRedirect()` as the default child-redirector function for any containers that won’t accept other widgets. `PtCompoundRedirect()` redirects child creation to the parent of the container widget. This causes a new widget to become a sibling of the container widget rather than a child of the container widget.

If you set up a child-redirector function in your widget class, `Pt_CHILD_REDIRECTOR` is set. When you create an instance, this bit is automatically turned off before Defaults method is called. It’s turned back on after the Defaults chaining is complete. This lets you create subordinate children in your Defaults method without having to clear and set this bit yourself.

### Fundamental methods

When using container widgets, you must provide a number of container-specific processes within the fundamental methods in addition to those common to every widget.

#### Defaults method

To enable the container-constraint methods, set the appropriate bits in `container->flags`. You can turn on all bits as follows:

```c
ctnr->flags |= Pt_CONTAINER_CONSTRAINT_BITS;
```

Only the constraint methods whose bits are on are invoked. If you turn a bit on and the corresponding constraint method is undefined, the bit is ignored. For example:

```c
ctnr->flags |= Pt_CHILD_CREATED;
```

For each bit set in the flag, you should provide a constraint method in the argument list of the class-creation function:

```c
static PtArg_t args[] = {
    ...
    { Pt_SET_CHILD_CREATED_F, child_created },
    ...
};
```

The `child_created()` function should modify the child or container as required to suit the situation.
The Common User Access (CUA) mechanism built into the Photon library automatically passes focus around the application. If you want to prevent widgets inside the container from getting focus, set the `Pt_BLOCK_CUA_FOCUS` flag. The `PtMenuBar` widget does this (i.e. you can't press the Tab key to move the focus from a widget outside a menubar to a widget inside a menubar).

**Extent method**

This method is responsible for determining the “screen real estate” of a widget. Anchoring is applied in this method. Widgets subclassed under `PtContainer` normally call the Extent method of `PtContainer` to do the final extent calculation and anchor the widget according to its anchor flags. When a widget’s extent has been calculated, its `widget->extent_valid` flag should be set to `Pt_TRUE`.

The following example demonstrates how to handle anchoring if you choose not to let the `PtContainer` class do it for you (see also `PtSuperClassExtent()`):

```c
mycontainerwidget_extent( PtWidget_t *widget )
{
    PtWidget_t *widget
    PhRect_t canvas, old_extent;
    PhArea_t area;

    // Store the old size for comparison later.
    old_extent = widget->extent;
    PhRectToArea( &old_extent, &area );

    if( PtResizePolicy( widget ) )
    {
        PhRect_t render;
        render.lr.x = render.lr.y = SHRT_MIN;
        render.ul.x = render.ul.y = SHRT_MAX;
        PtChildBoundingBox( widget, &canvas, &render );
        PhTranslateRect( &render, &canvas.ul );
        PtAttemptResize( widget, &render, &canvas );
    }
    PtSuperClassExtent( PtBasic, widget );
    widget->extent_valid = Pt_TRUE;

    // Containers must anchor their children. Flux to
    // minimize flicker.
    PtStartFlux( widget );
    if( widget->parent && widget->parent->extent_valid
        && (ctnr->anchor_flags & Pt_IS_ANCHORED )
        && !( widget->class_rec->flags &
            (Pt_DISJOINT|Pt_DISCONTINUOUS) ) )
        PtAnchorWidget( widget );
    else
        if( !(ctnr->anchor_flags & Pt_ANCHORS_LOCKED )
            && memcmp( &widget->area.size, &area.size,
                sizeof(PhDim_t) ) )
            for( wlp = ctnr->ctnrs; wlp; wlp = next )
            {
                next = wlp->next;
                if( ((PtContainerWidget_t *)wlp->widget)->anchor_flags &
                    Pt_IS_ANCHORED )
```
PtAnchorWidget( wlp->widget );

PtEndFlux( widget );

// If the size changes, accommodate the new size.
if( memcmp( &old_extent, &widget->extent, sizeof( old_extent ) ) )
{
  if( ( widget->flags & Pt_REALIZED ) && (widget->rid) )
    PtCoreChangeRegion( Ph_REGION_ORIGIN | Ph_REGION_RECT, widget );
  if( !(ctnr->anchor_flags & Pt_ANCHORS_LOCKED)
    && memcmp( &widget->area.size, &area.size, sizeof(PhDim_t) ) )
    PtInvokeResizeCallbacks( widget );
}

Realization method

If your container creates any subordinate widgets with the Pt_DELAY_REALIZE flag set, they can be realized in the Realization method.

Compound widget anatomy

This section applies to creating PtCompound class widgets:

PtWidget \rightarrow PtBasic \rightarrow PtContainer \rightarrow PtCompound \rightarrow MyCompound

The PtCompound superclass supports extended functionality provided by “exported” subordinate children. The export mechanism allows users to set/get the resources of subordinate children through a compound widget without defining any of these resources in the compound widget. Compound widgets can modify or block the default resources inherited by their exported subordinate children.

Widgets don’t have to be a subclass of PtCompound to export subordinate widgets. However, because PtContainer provides powerful child-constraint and child-redirector mechanisms, we recommend that if you’re building a widget that creates subordinate children, you should subclass that widget to PtContainer (if not PtCompound). This greatly simplifies the management of subordinate children. It also makes the widget easier to use since new resources don’t have to be learned.

To achieve this exporting mechanism, the compound class extends the PtContainer widget class definition as shown below:

typedef struct Pt_compound_class {
  PtContainerClass_t container;
  ushort_t num_subordinates;
  ushort_t *subordinates;
  ushort_t num_blocked_resources;
  ulong_t *blocked_resources;
} PtCompoundClass_t;

The members of this structure are:
num_subordinates (Resource manifest Pt_SET_NUM_SUBORDINATES)

The number of subordinate widgets (in the subordinate offset array) to be exported. This value must be supplied when defining an array of offsets.

subordinates (Resource manifest Pt_SET_SUBORDINATES)

An array of offsets to widget pointers to subordinate widgets. Every widget defined in the array of offsets must be created in the Defaults method of the compound widget.

num_blocked_resources (Resource manifest Pt_SET_NUM_BLOCKED_RESOURCES)

The number of resources in the array of blocked resources.

blocked_resources (Resource manifest Pt_SET_BLOCKED_RESOURCES)

An array of resources to be blocked.

Here’s a sample class-creation function from PtComboBox:

```c
// PtComboBox class creation function

static PtWidgetClass_t *PtCreateComboBoxClass( void )
{
    static const PtResourceRec_t resources[] = {
        { Pt_ARG_CURSOR_TYPE, combobox_modify, 0,
            PT_ARG_IS_NUMBER( PtWidget_t, cursor_type ) },
        { Pt_ARG_CURSOR_COLOR, combobox_modify, 0,
            PT_ARG_IS_NUMBER( PtWidget_t, cursor_color ) },
        { Pt_ARG_BORDER WIDTH, combobox_modify, 0,
            PT_ARG_IS_NUMBER( PtComboBoxWidget_t, border_width ) },
        { Pt_ARG_FLAGS, combobox_modify, 0,
            PT_ARG_IS_FLAGS( PtWidget_t, flags ) },
        { Pt_ARG_TOP_BORDER_COLOR, set_border_color, 0,
            PT_ARG_IS_NUMBER( PtBasicWidget_t, top_border_color ) },
        { Pt_ARG_HORIZON TAL_ALIGNMENT, combobox_modify, 0,
            PT_ARG_IS_NUMBER( PtLabelWidget_t, h_alignment ) },
        { Pt_ARG_VERTICAL_ALIGNMENT, combobox_modify, 0,
            PT_ARG_IS_NUMBER( PtLabelWidget_t, v_alignment ) },
        { Pt_ARG_MARGIN_WIDTH, combobox_modify, 0,
            PT_ARG_IS_NUMBER( PtBasicWidget_t, margin_width ) },
        { Pt_ARG_MARGIN_HEIGHT, combobox_modify, 0,
            PT_ARG_IS_NUMBER( PtLabelWidget_t, margin_height ) },
        { Pt_ARG_MARGIN_TOP, combobox_modify, 0,
            PT_ARG_IS_NUMBER( PtLabelWidget_t, margin_top ) },
        { Pt_ARG_MARGIN_BOTTOM, combobox_modify, 0,
            PT_ARG_IS_NUMBER( PtLabelWidget_t, margin_bottom ) },
        { Pt_ARG_COLOR, set_all_same_res, get_text_res,
            PT_ARG_IS_NUMBER( PtComboBoxUnion_t, basic.color ) },
        { Pt_ARG_FILL COLOR, combobox_modify, 0,
            PT_ARG_IS_COLOR( PtBasicWidget_t, fill_color ) },
        { Pt_ARG_TOP BORDER_COLOR, set_border_color, 0,
            PT_ARG_IS_NUMBER( PtBasicWidget_t, top_border_color ) },
    };
    return &resources;
}
```
{ Pt_ARG_IS_NUMBER( PtBasicWidget_t, bot_border_color ) },
{ Pt_ARG_TEXT_STRING, set_text_res, get_text_res },
{ Pt_ARG_TEXT_FLAGS, combobox_modify, get_text_res, Pt_ARG_IS_FLAGS( PtComboBoxWidget_t, flags ) },
{ Pt_ARG_CB_SELECTION_FILL_COLOR, set_list_res, get_list_res, Pt_ARG_IS_FLAGS( PtComboBoxWidget_t, flags ) },
{ Pt_ARG_CB_SELECTION, (long) Pt_CHANGE_INVISIBLE, 0, Pt_ARG_IS_IN_CB_LISTS( PtCallback_t ) },
{ Pt_ARG_CB_LIST_INPUT, (long) Pt_CHANGE_INVISIBLE, 0, Pt_ARG_IS_IN_CB_LISTS( PtCallback_t ) },
{ Pt_ARG_CB_CBOX_ACTIVATE, (long) Pt_CHANGE_INVISIBLE, 0, Pt_ARG_IS_IN_CB_LISTS( PtCallback_t ) },
{ Pt_ARG_CB_CBOX_BUTTON_BORDER_WIDTH, (long) Pt_CHANGE_RESIZE_REDRAW, 0, Pt_ARG_IS_FLAG( PtComboBoxWidget_t, butn_border_width ) },
{ Pt_ARG_CB_CBOX_BUTTON_TOP_BORDER_COLOR, (long) Pt_CHANGE_REDRAW, 0, Pt_ARG_IS_FLAG( PtComboBoxWidget_t, butn_top_border_color ) },
{ Pt_ARG_CB_CBOX_BUTTON_BOT_BORDER_COLOR, (long) Pt_CHANGE_REDRAW, 0, Pt_ARG_IS_FLAG( PtComboBoxWidget_t, butn_bot_border_color ) },
{ Pt_ARG_CB_CBOX_BUTTON_COLOR, (long) Pt_CHANGE_REDRAW, 0, Pt_ARG_IS_FLAG( PtComboBoxWidget_t, butn_color ) },
{ Pt_ARG_CB_CBOX_MAX_VISIBLE_COUNT, (long) Pt_CHANGE_INVISIBLE, 0, Pt_ARG_IS_FLAG( PtComboBoxWidget_t, max_visible ) },
{ Pt_CB_CBOX_CLOSE, (long) Pt_CHANGE_INVISIBLE, 0, Pt_ARG_IS_IN_CB_LISTS( PtCallback_t ) },
};

static const ushort_t subs[] = {
offsetof( PtComboBoxWidget_t, text_wgt ),
offsetof( PtComboBoxWidget_t, list_wgt ),
};

static const PtClassRawCallback_t callback = {
Ph_EV_KEY, combobox_callback
};

static const ulong_t blocked[] = {
Pt_ARG_SELECTION_MODE,
Pt_ARG_SEL_INDEXES,
Pt_ARG_COLUMNS,
};

static const PtWidgetAction_t actions[] = {
{ Ph_EV_BUT_PRESS | Ph_EV_BUT_RELEASE | Ph_EV_BUT_REPEAT, { combobox_but_action } },
};

static const PtClassArg_t args[] = {
{ Pt_SET_DFLTS_F, (void*) combobox_dflts },
{ Pt_SET_RAW_CALLBACKS, (void*) &callback },
{ Pt_SET_RESOURCES, (void*) resources },
{ Pt_SET_EXTRA_RESOURCES, extra_resources },
{ Pt_SET_EXTENT_F, (void*) combobox_extent },
{ Pt_SET_CONNECT_F, (void*) combobox_init },
};
In the example code shown above, the `PtComboBox` widget creates subordinate widgets and sets the pointers `text_wgt` and `list_wgt` in the Defaults method to the appropriate widget pointer.

In the following example, the widget class `MyCompound` creates subordinate widgets `PtText`, `PtProgress`, `PtScrollBarArea`, and a few `PtLabels`. The `PtText`, `PtProgress`, and `PtScrollBarArea` widgets are exported, but the `PtLabel` widgets aren’t. All resources are applied to the subordinate widgets. No extra code is required for `MyCompound`:

```
: 
: n = 0;
PtSetArg( &argt[n], Pt_ARG_POS, &pos, 0 ); n++;
PtSetArg( &argt[n], Pt_ARG_TEXT_STRING, "A text field", 0 ); n++;
PtSetArg( &argt[n], Pt_ARG_SCROLL_AREA_MAX_X, 1000, 0 ); n++;
PtSetArg( &argt[n], Pt_CB_SCROLLED_X, &callback, l ); n++;
PtSetResources( MyCompound, n, argt ); 
: 
: n = 0;
```

To get the resources afterwards:

```
: 
: n = 0;
n = 0;
PtSetArg( &argt[n], Pt_ARG_POS, &phpoint_ptr, 0 ); n++;
PtSetArg( &argt[n], Pt_ARG_TEXT_STRING, &char_ptr, 0 ); n++;
PtSetArg( &argt[n], Pt_ARG_SCROLL_AREA_MAX_X, &short_ptr, 0 ); n++;
```
When the **Pt_CB_SCROLLED_X** callback is invoked, the first parameter *widget* is a pointer to *MyCompound*, not to the subordinate. The *PtCompound* class also makes it easy to block specific resources from hitting any subordinate.

What happens when you export two or more subordinate widgets supporting the same resource? For example, both a button and a text field accept the resource **Pt_ARG_TEXT_STRING**; if they’re exported with no supporting code, then setting the resource on the compound widget will set the resource on both subordinates.

When getting the resource, the value is retrieved from the first exported widget supporting the resource. If you don’t want this to happen, set up a resource-redirector function at the end of the compound widget’s Defaults method.

You can have a compound widget override the resource (see **Pt_ARG_TEXT_STRING** in the example above) by defining it, then using the `mod_f()` function (named in the `mod_f` field member of the *PtResourceRec_t* structure) to apply the resource to the appropriate subordinate.

Any resources defined for a subclass of a *PtCompound* widget won’t automatically be applied to a subordinate widget, exported or otherwise. If you define resource-redirector functions for all user-modified resources of subordinates, consider subclassing your widget below *PtContainer* instead of *PtCompound*.

### Blocking resources

The exporting mechanism provides a resource-blocking mechanism. You can selectively block resources, making them inaccessible to subordinate widgets.

Suppose you’ve built a compound widget comprising an exported *PtSlider* widget, and you want the slider thumb to stay the same size always — blocking access to the **Pt_ARG_SLIDER_SIZE** resource would prevent the slider from being resized.

You may also find that you have more than one occurrence of a single type of widget as subordinate widget. For example, your widget might have a vertical and a horizontal scrollbar. In this case, you should block **Pt_ARG_SCROLL_POSITION** and create two new resources: **Pt_ARG_X_SCROLL_POSITION** and **Pt_ARG_Y_SCROLL_POSITION**. The Set Resources method for each would set the **Pt_ARG_SCROLL_POSITION** resource of the appropriate subordinate scrollbar widget.

### Compound redirection

The *PtCompound* class provides a standard child-redirector function called *PtCompoundRedirect()*. This function prevents the user from creating widgets in your compound widget and redirects the user’s widget to another container instead. To target another widget, redirect child creation in the class-creation function:

```c
{ Pt_SET_CHILD_REDIRECT_F, (long)PtCompoundRedirect },
```
In the template for the **PtSampCompound** widget, new children are redirected to one of the procreated subordinate widgets, *scroll_area*, by calling **PtValidParent()** (described in the Photon *Library Reference*):

```c
PtWidget_t * PtSampContainerRedirect( PtWidget_t *widget )
{
    PtWidget_t *parent;
    if( ( parent = PtValidParent( samp->scroll_area,
                                 widget->class_ref ) ) == widget )
        return PtWidgetParent( widget );
    return( parent );

    /*
    * Returning samp->scroll_area would allow the child
    * to be created as a direct child of samp->scroll_area.
    * This may be undesirable, as scroll_area is a container
    * widget that redirects its children.
    */
}
```

**PtValidParent()** honors any child-redirector functions existing in subordinate widgets and their subordinates. **PtValidParent()** should be used only to redirect parentage to a subordinate child. This prevents infinite loops when the child’s redirector function returns control.

**Fundamental methods**

Compound widgets require a few fundamental methods in addition to those common to every widget.

**Defaults method**

For compound widgets, all exported subordinate widgets *must* be created in the Defaults method. To export widgets, set the following in the class-creation function:

```c
Pt_SET_SUBORDINATES
    An array of offsets to the subordinate widget pointers.

Pt_SET_NUM_SUBORDINATES
    The number of entries in the array.
```

Resources of subordinate widgets may be overridden, overloaded, or blocked. Subordinate widgets can be created but not realized in the Defaults method.

For the container-constraint mechanisms to work correctly, the **Pt_PROCREATED** flag has to be set in each subordinate widget, and the **Pt_ARG_DATA (not Pt_ARG_USER_DATA)** resource has to point back to the parent widget. Otherwise, you’ll be greeted with a SIGSEGV error.
Your custom widget shouldn’t set its own \texttt{Pt\_ARG\_DATA} resource because this resource is used internally by the Photon libraries. It is safe to set it in your widget’s subordinate children.

Here’s an example from \texttt{PtComboBox}:

```c
static void combobox_dflts( PtWidget_t *widget )
{
    PtComboBoxUnion_t *combobox = (PtComboBoxUnion_t *)widget;
    PtArg_t args[13];
    int n = 0;
    PtCallback_t callback;
    PtRawCallback_t raw_cb;

    combobox->core.flags |= Pt_HIGHLIGHTED | Pt_SET;
    combobox->core.flags &= ~Pt_GETS_FOCUS;
    combobox->core.resize_flags = Pt_RESIZE_XY_ALWAYS;
    combobox->basic.fill_color = Pg_LGREY;
    combobox->basic.margin_height = 0;
    combobox->basic.margin_width = 0;
    combobox->basic.flags = Pt_STATIC_GRADIENT | Pt_ALLETCHES | Pt_ALLOUTLINES;
    widget->border_width = combobox->combobox.border_width = 2;
    combobox->combobox.butn_size.w = 13;
    combobox->combobox.butn_border_color = Pg_DGREY;
    combobox->combobox.butn_top_border_color = Pg_WHITE;
    combobox->combobox.butn_color = Pg_GREY;
    combobox->combobox.flags = Pt_COMBOBOX_DAMAGE_BUTTON;
    callback.event_f = combobox_text_callback;
    callback.data = (void*)widget;

    n = 0;
    PtSetArg( &args[n], Pt_ARG_BEVEL_WIDTH, 0, 0 );
    n++;
    PtSetArg( &args[n], Pt_ARG_MARGIN_HEIGHT, 2, 0 );
    n++;
    PtSetArg( &args[n], Pt_ARG_MARGIN_WIDTH, 2, 0 );
    n++;
    PtSetArg( &args[n], Pt_ARG_DATA, &widget, sizeof( widget ) );
    n++;
    PtSetArg( &args[n], Pt_ARG_FLAGS, Pt_PROCREATED, Pt_PROCREATED );
    n++;
    PtSetArg( &args[n], Pt_CB_MODIFY_VERIFY, &callback, 1 );
    n++;
    PtSetArg( &args[n], Pt_CB_MOTION_VERIFY, &callback, 1 );
    n++;
    PtSetArg( &args[n], Pt_CB_ACTIVATE, &callback, 1 );
    n++;
    PtSetArg( &args[n], Pt_CB_GOT_FOCUS, &callback, 1 );
    n++;
    PtSetArg( &args[n], Pt_CB_LOST_FOCUS, &callback, 1 );
    n++;
    PtSetArg( &args[n], Pt_ARG_RESIZE_FLAGS, Pt_RESIZE_Y AS REQUIRED,
              Pt_RESIZE_XY_BITS );
    n++;
    PtSetArg( &args[n], Pt_ARG_BASIC_FLAGS, Pt_TOP_LEFT_INLINE
              | Pt_RIGHT_OUTLINE | Pt_FLAT_FILL, "0" );
    n++;
    combobox->combobox.text_wgt =
        PtCreateWidget( PtText, widget, n, args );

    n = 0;
    callback.event_f = combobox_list_callback;
    callback.data = (void*)widget;
    raw_cb.event_f = combobox_action;
    raw_cb.event_mask = Ph_EV_BOUNDARY;
    raw_cb.data = combobox;
    PtSetArg( &args[n], Pt_ARG_BEVEL_WIDTH, 1, 0 );
    n++;
    PtSetArg( &args[n], Pt_ARG_SCROLLBAR_WIDTH,
              combobox->combobox.butn_size.w +3, 0 );
```

Chapter 3 • Anatomy of a Widget

September 10, 2007
n++;  
PtSetArg( &args[n], Pt_ARG_SEL_MODE, Pt_SELECTION_MODE_SINGLE  
| Pt_SELECTION_MODE_AUTO, 0 );

n++;  
PtSetArg( &args[n], Pt_ARG_FLAGS, Pt_PROCREATED | Pt_DELAY_REALIZE,  
Pt_GETS_FOCUS | PtETCH_HIGHLIGHT  
| Pt_PROCREATED | Pt_DELAY_REALIZE | PtETCH_HIGHLIGHT);  

n++;  
PtSetArg( &args[n], Pt_CB_SELECTION, &callback, 1 );

n++;  
PtSetArg( &args[n], Pt_CB_LIST_INPUT, &callback, 1 );

n++;  
PtSetArg( &args[n], Pt_CB_RAW, &raw_cb, 1 );

n++;  
PtSetArg( &args[n], Pt_ARG_DATA, &widget,  
sizeof( widget ) );

n++;  
PtSetArg( &args[n++], Pt_CB_ARM, &callback, 1 );

n = 0;

combobox->list_wgt = PtCreateWidget( PtList, widget,  
n, args );

Realization method

The Realization method is the last class-member function called before the Draw  
method. It’s where you realize subordinate widgets created in the Defaults method.  
Here’s an example from PtComboBox:

static int combobox_realized( PtWidget_t *widget )  
{
  PtComboBoxWidget_t *combobox = (PtComboBoxWidget_t *)widget;
  long flags;

  if ( combobox->flags & Pt_COMBOBOX_STATIC )
    PtRealizeWidget( combobox->list_wgt );
  else {
    PtRealizeWidget( combobox->list_wgt );
  }

  return Pt_CONTINUE;
}

Get Resources and Set Resources methods

To date, the PtCompound widget class is the only class to redefine the Get Resources  
and Set Resources methods, which are used internally. You don’t have to set a function  
for this method unless you intend to duplicate the behavior of PtCompound.
**Destruction method**

The Destruction method must destroy any redirected callback lists of exported subordinates having callback resources set on them.
Chapter 4
Using Widget Superclasses

In this chapter...

PtWidget  73
PtBasic    77
PtContainer 81
PtCompound 87
PtGenList  89
PtGenTree  96
PtLabel  100
PtGraphic 103
PtGauge  105
This chapter describes the common widget superclasses. Use this information to determine whether a superclass’s method does what you need for your widget. With this knowledge you can decide whether to take over the method (override inheritance), add a method to the chain, or use the \textit{PtSuperClass*} functions.

It’s important to read the specifications of each of your widget’s superclasses before using any of the \textit{PtSuperClass*()} functions.

\textbf{PtWidget}

The \texttt{PtWidget} superclass provides the fundamental functionality of all Photon widgets.

\textbf{Class hierarchy}

\texttt{PtWidget}

\textbf{Class flags}

\texttt{Pt\_RECTANGULAR}

\textbf{Methods}

\texttt{PtWidget} defines the class methods described below.

\textbf{Defaults method}

None.

\textbf{Initialization method}

None.

\textbf{Extent method}

Calculates the extent of the widget without accounting for borders or margins. The extent is calculated based on the widget’s position (\texttt{Pt\_ARG\_POS}) and dimension (\texttt{Pt\_ARG\_DIM}). For disjoint widgets (e.g. \texttt{PtWindow} widgets), the position is assumed to be (0, 0).

\textbf{Connection method}

Determines if the widget needs a region, and if so, what events the region should be opaque and/or sensitive to.

If the widget has a bitmap cursor defined, it’s set up as region data. If the widget already has a region, that region is updated with the opaque/sense and region data. This is done through \texttt{PhRegionChange()}.

If the widget doesn’t have a region but one is required, it’s created through \texttt{PhRegionOpen()}. 
Finally, if the widget isn’t a window, any hotkey callbacks attached to the widget are registered with the nearest disjoint parent (of PtWindow class). This is done through PtSetResources().

Realization method

None.

Unrealization method

If the widget has a region, the region is closed. All descendants (widget->rid members) are set to 0. If the widget isn’t a window, any attached hotkey callbacks are detached from its disjoint parent.

The extent of this widget is damaged on the parent. All children of this widget are unrealized. Any of these widgets having a constraining parent with Pt_SET_CHILD_UNREALIZED_F defined has that function invoked. Any of these having a Pt_CB_UNREALIZED callback chain has that callback chain invoked (from the top down).

Destruction method

Called when a widget instance is being removed. Frees widget->widget_data. Any memory automatically allocated by a resource is released (this memory shouldn’t be freed in the Destruction method). Any resources you free must have the pointer set to NULL to prevent the widget engine from freeing the memory a second time. The following resource types automatically allocate and release memory:

- Pt_ARG_IS_ALLOC
- Pt_ARG_IS_ARRAY
- Pt_ARG_IS_CALLBACK_LIST
- Pt_ARG_IS_LINK
- Pt_ARG_IS_STRING

Widget actions

Raw callbacks are supported for Ph_EV_BUT_PRESS, Ph_EV_BUT_RELEASE, Ph_EV_BUT_REPEAT, and Ph_EV_KEY events.

If there are callbacks in the Pt_CB_FILTER callback chain having an event mask that matches the current event type, those callbacks are invoked. If the value returned by a filter callback is greater than 0, that value is returned. If there are balloons registered with this widget, the balloon list is checked to determine if a balloon needs to be inflated and/or deflated.

If the event is a Ph_EV_KEY event, the event is given to the focused child of the widget, if there is one. If the event isn’t consumed by the widget, the hotkey chain of the nearest disjoint parent is processed to see if the key is a hotkey. If a matching
If the event is a button press, release, or repeat, each child widget intersecting the event is found via `PtContainerHit()` until one of these widgets consumes the event.

When an intersecting widget is found, the event rectangle is translated relative to the upper-left corner of the intersecting widget’s canvas. Once translation has been performed, the event is given to the intersecting widget via `absorbed = PtEventHandler()`. Upon return, the event rectangle is detranslated by the same amount.

If all the following are true:

- the event was a button press
- the event hasn’t yet directed focus
- the current intersecting widget has the Pt_GETS_FOCUS flag set

then focus is given to that widget and `event->processing_flags` has its Ph_DIRECTED_FOCUS bit set. If the return from `PtEventHandler()` was Pt_END, the event is considered consumed and Pt_END is returned.

An extra step is performed if the event is a press and `_Pt->current` is NULL: the absorbing widget is recorded for the purposes of delivering phantom releases to the appropriate widget at a later time.

```c
if( widget->rid )
    //if the event was not absorbed, take it back.
    cbinfo->event->collector.rid = widget->rid;

if( ( ( absorbed != Pt_CONTINUE ) ||
    ( cbinfo->event->processing_flags & Ph_BACK_EVENT ) )
    && wp && cbinfo->event->type == Ph_EV_BUT_PRESS
    && ( !_Pt->current || _Pt->current == wp ) )
{
    if ( wp->flags & Pt_DESTROYED )
        _Pt->current = (PtWidget_t *)Pt_END;
    else
        _Pt->current = wp;
}
```

Once the event has been consumed or there are no more intersecting events, Pt_END is returned. Otherwise, `absorbed` is returned.

### Resource definitions

```c
static const PtResourceRec_t resources[] = {
    { Pt_ARG_AREA,          Pt_CHANGE_CANVAS,  0,
      Pt_ARG_IS_STRUCT( PtWidget_t, area )},
    { Pt_ARG_BEVEL_WIDTH,   Pt_CHANGE_CANVAS_REDRAW, 0,
      Pt_ARG_IS_NUMBER( PtWidget_t, border_width ) },
    { Pt_ARG_CURSOR_TYPE,   core_modify_cursor, 0,
      Pt_ARG_IS_NUMBER( PtWidget_t, cursor_type )},
    { Pt_ARG_CURSOR_COLOR,  core_modify_cursor, 0,
      Pt_ARG_IS_NUMBER( PtWidget_t, cursor_color )},
    { Pt_ARG_DATA,          Pt_CHANGE_INVISIBLE, 0,
```
Functions

```
static core_modify_cursor()
```

Sets the resource via direct assignment and if realized, sets the Ph_REGION_CURSOR bit in widget->flags. If this flag is on when the PtUpdate() function is called (see the Photon Library Reference), a PhRegionChange() will be performed to change the region’s cursor or create a region if one doesn’t exist.


```c
static core_modify_raw_callbacks()

Sets the resource via PtSetStruct() and adjusts the widget’s region sensitivity.

static core_modify_hotkey_callbacks()

Sets the resource via PtSetStruct() and if realized, attaches any hotkey callbacks of a nonwindow widget to its disjoint parent.

int PtModifyFlags()

Knocks down the read-only flag bits from the mask and applies the remaining flag to widget->flags. A change in the following flag bits may cause the widget to change:

Pt_AUTOHIGHLIGHT
   Requires a region and sensitivity to Ph_EV_BOUNDARY events via PhRegionChange() or the Connection method of PtWidget.

PtETCH_HIGHLIGHT
   Resize by setting the Pt_WIDGET_RESIZE bit in widget->flags.

Pt_SET
   If Pt_SELECT_NOREDRAW isn’t set in the widget flags, the widget will force a redraw by calling PtDamageWidget().

Pt_HIGHLIGHTED
   The widget will force a redraw by calling PtDamageWidget().
```

PtBasic

Provides fundamental Photon widget behavior and callbacks.

Class hierarchy

PtWidget → PtBasic

Class extensions

The PtBasic widget adds three class-level methods to fundamental widget functionality. For complete descriptions, see “Class methods” in the Anatomy of a Widget chapter.

```c
typedef struct Pt_basic_widget_class {
   PtWidgetClass_t core;
   void (*got_focus_f)( PtWidget_t *, PhEvent_t * );
   void (*lost_focus_f)( PtWidget_t *, PhEvent_t * );
   void (*calc_opaque_f)( PtWidget_t * );
} PtBasicWidgetClass_t;

#define Pt_SET_GOT_FOCUS_F (Pt_ARG_IS_POINTER(PtBasicWidgetClass_t,got_focus_f))
#define Pt_SET_LOST_FOCUS_F (Pt_ARG_IS_POINTER(PtBasicWidgetClass_t,lost_focus_f))
#define Pt_SET_CALC_OPAQUE_F (Pt_ARG_IS_POINTER(PtBasicWidgetClass_t,calc_opaque_f))
```
Methods

**PtBasic** defines the class methods described below.

**Defaults method**

```c
widget->border_width = 1;
widget->cursor_color = Ph_CURSOR_DEFAULT_COLOR;
widget->resize_flags = PT_RESIZE_XY_AS_REQUIRED;
widget->eflags = Pt_DAMAGE_ON_FOCUS;

basic->border_color = basic->fill_color = Pg_LGREY;
basic->flags = Pt_ALL_ETCHES | Pt_ALL_BEVELS |
                Pt_ALL_OUTLINES | Pt_FLAT_FILL;
basic->inline_color = basic->outline_color = PgGrey( 0x4b );
```

**Initialization method**
None.

**Extent method**

Invokes the Extent method of **PtWidget** via **PtSuperClassExtent()**. The resulting extent is then adjusted to account for borders and margins.

The Extent method calculates the widget’s opaque rectangle and determines whether the widget’s Pt_OPAQUE flag is set. The opaque rectangle, `widget->opaque_rect`, indicates:

- the rectangle under which damage needn’t occur if this widget is damaged
- what should be damaged if this widget changes position or dimension

This rectangle is the area capable of obscuring any widgets beneath. Widgets completely obscured by another widget aren’t drawn.

**Connection method**

Checks the **PtBasic** class structure member `calcOpaqueF`. If not NULL, the function is invoked. The `calcOpaqueF` function is responsible for setting or clearing a widget’s Pt_OPAQUE flag. If the widget has a rectangular area blocking anything beneath, the Pt_OPAQUE flag should be set and `widget->opaque_rect` should identify that rectangular area. Otherwise, the Pt_OPAQUE flag should be cleared.

If a widget’s fill color is Pg_TRANSPARENT or has a `basic->roundness` that’s greater than 0, the widget shouldn’t be flagged as opaque.

**Realization method**

Inherited from **PtWidget**.
Draw method

PtBasic’s Draw method draws a rectangle filled with the current fill color. The widget’s border is rendered if the Pt_HIGHLIGHTED bit of the widget flags is set and the border width is greater than 0.

If the Pt_SET bit is set, the border is rendered inverted (i.e. the top border color is used to draw the bottom border and vice versa). PtBasic’s Draw method also handles focus rendering.

Unrealization method

None.

Destruction method

None.

Got Focus method

Damages widgets having both the Pt_FOCUS_RENDER and Pt_DAMAGE_ON_FOCUS flags set. Highlights and invokes the basic->arm (Pt_CB_ARM) callback list if this widget has the Pt_AUTOHIGHLIGHT flag set. Invokes the basic->got_focus (Pt_CB_GOT_FOCUS) callback list using PtInvokeCallbackList().

Lost Focus method

Damages widgets having both the Pt_FOCUS_RENDER and Pt_DAMAGE_ON_FOCUS flags set. Unhighlights and invokes the basic->disarm (Pt_CB_DISARM) callback list if this widget has the Pt_AUTOHIGHLIGHT flag set. Invokes the basic->got_focus (Pt_CB_LOST_FOCUS) callback list via PtInvokeCallbackList().

Calc Opaque Rect method

Sets or clears the widget’s Pt_OPAQUE flag (Pt_ARG_FLAGS resource) based on the widget’s fill color, roundness, and current value of the Pt_RECTANGULAR flag in the widget class structure.

Widget actions

PtBasic is sensitive to the following events:

Ph_EV_BUT_PRESS

The widget sets/clears Pt_SET flags, invokes Pt_CB_ARM (and possibly Pt_CB_ACTIVATE if a toggle button is selected) and Pt_CB_MENU (if the right mouse button is pressed).

Ph_EV_BUT_REPEAT

The widget invokes the Pt_CB_REPEAT callbacks.
The widget invokes the `Pt_CB_DISARM` callbacks. If a real release (as opposed to a phantom release) is received, the widget also invokes `Pt_CB_ACTIVATE`.

The widget manages highlighting of widgets with the `Pt_AUTOHIGHLIGHT` flag set.

The raw callbacks return `Pt_CONTINUE` to allow chaining to continue.

---

If your widget class defines `Pt_SET_RAW_CALLBACKS`, keep this in mind: if your raw callback is sensitive to one of the events listed above and the callback returns `Pt_END` or `Pt_HALT`, `PtBasic` behavior for that event won’t occur unless you invoke `PtBasic`’s raw handlers within the callback prior to returning. This can be done via `PtSuperClassRawEvent()`.

---

**Resource definitions**

```c
static const PtResourceRec_t resources[] = {
    { Pt_ARG_FLAGS, basic_modify_flags, NULL, 
      Pt_ARG_IS_FLAGS( PtWidget_t, flags ) },
    { Pt_ARG_INLINE_COLOR, Pt_CHANGE_INVISIBLE, NULL, 
      Pt_ARG_IS_COLOR( PtBasicWidget_t, inline_color ) },
    { Pt_ARG_COLOR, Pt_CHANGE_REDRAW, NULL, 
      Pt_ARG_IS_COLOR( PtBasicWidget_t, color ) },
    { Pt_ARG_FILL_COLOR, Pt_CHANGE_INVISIBLE, NULL, 
      Pt_ARG_IS_COLOR( PtBasicWidget_t, fill_color ) },
    { Pt_ARG_FILL_PATTERN, Pt_CHANGE_REDRAW, NULL, 
      Pt_ARG_IS_STRUCT( PtBasicWidget_t, fill_pattern ) },
    { Pt_ARG_MARGIN_HEIGHT, Pt_CHANGE_CANVAS, NULL, 
      Pt_ARG_IS_NUMBER( PtBasicWidget_t, margin_height ) },
    { Pt_ARG_MARGIN_WIDTH, Pt_CHANGE_CANVAS, NULL, 
      Pt_ARG_IS_NUMBER( PtBasicWidget_t, margin_width ) },
    { Pt_ARG_OUTLINE_COLOR, Pt_CHANGE_INVISIBLE, NULL, 
      Pt_ARG_IS_COLOR( PtBasicWidget_t, outline_color ) },
    { Pt_CB_ARM, Pt_CHANGE_INVISIBLE, NULL, 
      Pt_ARG_IS_CALLBACK_LIST( PtBasicWidget_t, arm ) },
    { Pt_CB_DISARM, Pt_CHANGE_INVISIBLE, NULL, 
      Pt_ARG_IS_CALLBACK_LIST( PtBasicWidget_t, disarm ) },
    { Pt_CB_ACTIVATE, Pt_CHANGE_INVISIBLE, NULL, 
      Pt_ARG_IS_CALLBACK_LIST( PtBasicWidget_t, activate ) },
    { Pt_CB_GOT_FOCUS, Pt_CHANGE_INVISIBLE, NULL, 
      Pt_ARG_IS_CALLBACK_LIST( PtBasicWidget_t, got_focus ) },
    { Pt_CB_LOST_FOCUS, Pt_CHANGE_INVISIBLE, NULL, 
      Pt_ARG_IS_CALLBACK_LIST( PtBasicWidget_t, lost_focus ) },
    { Pt_CB_REPEAT, Pt_CHANGE_INVISIBLE, NULL, 
      Pt_ARG_IS_CALLBACK_LIST( PtBasicWidget_t, repeat ) },
    { Pt_ARG_TRANS_PATTERN, Pt_CHANGE_RESIZE_REDRAW, NULL, 
      Pt_ARG_IS_STRUCT( PtBasicWidget_t, trans_pattern ) },
    { Pt_ARG_BASIC_FLAGS, basic_modify_flags, NULL, 
      Pt_ARG_IS_FLAGS( PtBasicWidget_t, flags ) },
    { Pt_CB_MENU, Pt_CHANGE_INVISIBLE, NULL, 
      Pt_ARG_IS_CALLBACK_LIST( PtBasicWidget_t, menu ) },
    { Pt_ARG_CONTRAST, Pt_CHANGE_INVISIBLE, NULL, 
      Pt_ARG_IS_NUMBER( PtBasicWidget_t, contrast ) },
    { Pt_ARG_BEVEL_CONTRAST, Pt_CHANGE_INVISIBLE, NULL, 
      Pt_ARG_IS_NUMBER( PtBasicWidget_t, bevel_contrast ) },
};
```
PtContainer

Functions

static void basic_modify_flags()

For $P t\_A R G\_F L A G S$, this function:

- recalculates the opaque rectangle if a highlighting flag is changing
- damages the widget if the $P t\_G H O S T$ flag is changing
- lets any other flag changes be handled via $P tS u p e rC l a s s S e tR e s o u r c e s()$

void basic_modify()

Sets the fill color and roundness values. Invokes the Connection method of $P tB a s i c$ (which invokes the Calc Opaque Rect method), and reexamines the widget’s clear/opaque status via $P tU p d a t eV i s i b i l i t y()$.

Setting the $P t\_C O N S U M E\_E V E N T S$ bit of $P tW i d g e t$’s $e f l a g s$ resource controls whether a widget consumes any event it’s given. Container widgets set this bit by default to prevent events from passing through to the subclassed widgets beneath them.

PtContainer

This superclass provides a new origin, clipping, and constraints for widget children. If you’re creating a widget with children, we recommend you make your widget a subclass of $P tC o n t a i n e r$ or one of its subclasses.

A container can have its own Draw method to render any widget-specific data (other than children).

Class hierarchy

$P tW i d g e t \rightarrow P tB a s i c \rightarrow P tC o n t a i n e r$
Class extensions

typedef struct Pt_container_widget_class {
  PtBasicWidgetClass_t basic;
  void (*child_created_f)(PtWidget_t *widget,
                           PtWidget_t *child);
  int (*child_settingresource_f)(PtWidget_t *widget,
                                 PtWidget_t *child,
                                 PtArg_t const *argt);
  int (*child_gettingresource_f)(PtWidget_t *widget,
                                 PtWidget_t *child,
                                 PtArg_t *argt);
  int (*child_realizing_f)(PtWidget_t *widget,
                           PtWidget_t *child);
  void (*child_realized_f)(PtWidget_t *widget,
                           PtWidget_t *child);
  void (*child_unrealizing_f)(PtWidget_t *widget,
                           PtWidget_t *child);
  void (*child_unrealized_f)(PtWidget_t *widget,
                           PtWidget_t *child);
  void (*child_destroyed_f)(PtWidget_t *widget,
                           PtWidget_t *child);
  void (*child_move_resize_f)(PtWidget_t *widget,
                           PtWidget_t *child,
                           PhArea_t *current_area,
                           PhRect_t *current_extent,
                           PhArea_t *old_area,
                           PhRect_t *old_extent);
  int (*child_getting_focus_f)(PtWidget_t *widget,
                           PtWidget_t *child,
                           PhEvent_t *event);
  int (*child_losing_focus_f)(PtWidget_t *widget,
                           PtWidget_t *child,
                           PhEvent_t *event);
  PtWidget_t *(*child_redirect_f)(PtWidget_t *,
                                   PtWidgetClassRef_t *);
} PtContainerClass_t;

#define Pt_SET_CHILD_SETTINGRESOURCE_F
     (Pt_ARG_MODE_PTR|offsetof(PtContainerClass_t,
                           child_settingresource_f))
#define Pt_SET_CHILD_GETTINGRESOURCE_F
     (Pt_ARG_MODE_PTR|offsetof(PtContainerClass_t,
                           child_gettingresource_f))
#define Pt_SET_CHILD_REALIZING_F
     (Pt_ARG_MODE_PTR|offsetof(PtContainerClass_t,
                           child_realizing_f))
#define Pt_SET_CHILD_REALIZED_F
     (Pt_ARG_MODE_PTR|offsetof(PtContainerClass_t,
                           child_realized_f))
#define Pt_SET_CHILD_UNREALIZING_F
     (Pt_ARG_MODE_PTR|offsetof(PtContainerClass_t,
                           child_unrealizing_f))
#define Pt_SET_CHILD_UNREALIZED_F
     (Pt_ARG_MODE_PTR|offsetof(PtContainerClass_t,
                           child_unrealized_f))
#define Pt_SET_CHILD_CREATED_F
     (Pt_ARG_MODE_PTR|offsetof(PtContainerClass_t,
                           child_created_f))
#define Pt_SET_CHILD_DESTROYED_F
     (Pt_ARG_MODE_PTR|offsetof(PtContainerClass_t,
                           child_destroyed_f))
#define Pt_SET_CHILD_MOVED_RESIZED_F

Methods

PtContainer defines the class methods described below.

Defaults method

```c
PtContainerWidget_t *ctnr = (PtContainerWidget_t *)widget;
PtBasicWidget_t *basic = (void *)widget;
widget->eflags |= Pt_CONSUME_EVENTS;
widget->border_width = 0;
widget->resize_flags &= ~Pt_RESIZE_XY_BITS;
basic->flags = Pt_ALL_BEVELS | Pt_FLAT_FILL;
ctnr->flags = Pt_ANCHOR_CHILD_HORIZONTAL |
               Pt_ANCHOR_CHILD_VERTICAL |
               Pt_CANVAS_INVALID | Pt_ENABLE_CUA |
               Pt_ENABLE_CUA_ARROWS;

ctnr->title_font = strdup("TextFont09");

PtSetParentWidget( widget );
```

Initialization method

Registers with its parent for anchoring services.

Extent method

Finds the bounding box of widget children and applies its resize policy. The extent is calculated, and if the canvas is different as a result, all registered child containers are anchored. If the extent or canvas is different, the resize callback list of the widget is invoked. See `PtSuperClassExtent()` for a sample Extent method of a container.

Connection method

None.

Realization method

None.
Draw method

Inherited from PtBasic.

Unrealization method

Deregisters a widget from its parent. Destroys the current_balloon and sets it to NULL (if not NULL already).

Destruction method

Deregisters a widget from its parent.

Got Focus method

Inherited from PtBasic.

Lost Focus method

Inherited from PtBasic.

Calc Opaque Rect method

Inherited from PtBasic.

Child Created method

None.

Child Realized/Unrealized method

```c
static void container_child_realized_unrealized(
    PtWidget_t *widget, PtWidget_t *child )
{
    PtContainerWidget_t *ctnr =
        (PtContainerWidget_t *)widget;
    if( child->flags & Pt_PROCREATED )
        // || !PtResizePolicy( widget )
        return;
    if ( ( widget->flags & Pt_REALIZED ) &&
        ( ctnr->flags & Pt_AUTO_EXTENT ) )
    {
        ctnr->flags |= Pt_IGNORE_CONSTRAINTS;
        PtMoveResizeWidget( widget, 0 );
        ctnr->flags &= ~Pt_IGNORE_CONSTRAINTS;
    }
}
```

Child Moved/Resized method

 Calls the Child Realized/Unrealized method with the correct parameters:

```c
static void container_child_moved_resized(
    PtWidget_t *widget, PtWidget_t *child, 
    PhArea_t *area, PhRect_t *rect, 
    PhArea_t *oarea, PhRect_t *orect )
{
    container_child_realized_unrealized( widget, child );
}
```
Child Destroyed method
None.

Child Setting Resources method
None.

Child Getting Resources method
None.

Child Getting Focus method
None.

Child Losing Focus method
None.

Widget actions
None.

Resource definitions

```c
static const PtRawCallback_t callback =
{
    Pt_EV_REDIRECTED, container_callback, NULL
};

static const PtResourceRec_t resources[] =
{
    { Pt_ARG_AREA, container_modify_area, NULL,
        Pt_ARG_IS_STRUCT( PtWidget_t, area ) },
    { Pt_ARG_DIM, container_modify_area, NULL,
        Pt_ARG_IS_STRUCT( PtWidget_t, area.size ) },
    { Pt_ARG_POS, container_modify_area, NULL,
        Pt_ARG_IS_STRUCT( PtWidget_t, area.pos ) },
    { Pt_ARG_RESIZE_FLAGS, container_modify_area, NULL,
        Pt_ARG_IS_FLAGS( PtWidget_t, resize_flags ) },
    { Pt_ARG_ANCHOR_OFFSETS, container_modify_area, NULL,
        Pt_ARG_IS_STRUCT( PtContainerWidget_t, anchor_offset ) },
    { Pt_ARG_ANCHOR_FLAGS, container_modify_area, NULL,
        Pt_ARG_IS_FLAGS( PtContainerWidget_t, anchor_flags ) },
    { Pt_ARG_FOCUS, container_modify, NULL,
        Pt_ARG_IS_POINTER( PtContainerWidget_t, focus ) },
    { Pt_CB_RESIZE, Pt_CHANGE_INVISIBLE, NULL,
        Pt_ARG_IS_CALLBACK_LIST( PtContainerWidget_t, resize ) },
    { Pt_CB_BALLOONS, container_set_balloons, NULL,
        Pt_ARG_IS_LINK( PtContainerWidget_t, balloons ) },
    { Pt_ARG_CONTAINER_FLAGS, container_modify_flags, NULL,
        Pt_ARG IS_FLAGS( PtContainerWidget_t, flags ) },
    { Pt_ARG_TITLE, container_set_title, NULL,
        Pt_ARG IS_STRING( PtContainerWidget_t, title ) },
    { Pt_ARG_TITLE_FONT, container_set_title, NULL,
        Pt_ARG IS_STRING( PtContainerWidget_t, title_font ) },
    { Pt_CB_CHILD_ADDED_REMOVED, Pt_CHANGE_INVISIBLE, NULL,
        Pt_ARG IS_LINK( PtContainerWidget_t, child_added_removed ) },
};
```
Functions

container_modify_area()

If area, pos, or dim is being modified, the container’s anchors are invalidated and unlocked:

container->flags |= Pt_ANCHORS_INVALID;
container->flags &= ~Pt_ANCHORS_LOCKED;

The Pt_ANCHORS_INVALID bit indicates that the widget can’t be anchored using the values in the anchor offsets member until these values are recalculated.

The Pt_ANCHORS_LOCKED bit indicates that the anchor offsets for the container are valid and shouldn’t be recalculated due to the change that just occurred. This bit is set when Pt_ARG_ANCHOR_OFFSETS is set. If Pt_ARG_ANCHOR_OFFSETS is set, the anchors are validated and locked. If Pt_ARG_RESIZE_FLAGS is being modified, the anchors are invalidated and the widget is flagged for resize:

widget->flags |= Pt_WIDGET_RESIZE;

container_modify()

static void container_modify( PtWidget_t *widget,
 PtArg_t *arg )
{

 PtContainerWidget_t *container =
 (PtContainerWidget_t *)widget;
 PhEvent_t event;
 memset( &event, 0, sizeof( event ) );
 switch( arg->type )
 {
 case Pt_ARG_FOCUS:
   // If the widget is already focused or isn’t
   // the immediate child of this container,
   // do nothing.
   if (( container->focus ==
       (PtWidget_t *)arg->value )
       || (!arg->value )
       || (((PtWidget_t *)arg->value)->parent !=
            widget))
     return;

   // otherwise, give the target widget focus.
   PtContainerGiveFocus( (PtWidget_t *)arg->value,
                        &event );
   if ( container->focus && container->focus->flags & Pt_REALIZED )
     PtDamageWidget( container->focus );

   
   }
}

container_set_balloons()

Registers/deregisters balloons with the container and increments/decrements the container’s balloon count (container->balloon_count). If the balloon count is
nonnull, the balloons_active bit located in container->anchor_flags is set; otherwise, it’s cleared.

A raw callback function to check the balloon list is added if the balloon count is nonzero and the balloons_active bit was not set. The callback is removed if the balloon count goes to zero and the balloons_active bit was set.

container Modify Flags

Container flags are set/cleared. If one of the bits affected was the Pt_AUTO_EXTENT bit, several constraint functions are enabled or disabled and the widget is (potentially) scheduled to have its Extent method executed.

```c
static void container_modify_flags( PtWidget_t *widget, PtArg_t const *argt )
{
    PtContainerWidget_t *ctnr = (PtContainerWidget_t *)widget;
    int changed;
    unsigned long mask = argt->len & ~Pt_CONTAINER_RO_FLAGS;
    changed = ctnr->flags;
    changed ^= ctnr->flags =
    ( ctnr->flags & ~mask ) |
    ( argt->value & mask );
    if( changed & Pt_AUTO_EXTENT )
    {
        if( ctnr->flags & Pt_AUTO_EXTENT )
        {
            ctnr->flags |= Pt_CHILD_REALIZED |
            Pt_CHILD_UNREALIZED |
            Pt_CHILDMOVED_RESIZED;
            widget->flags |= Pt_WIDGET_RESIZE;
            PtSyncWidget( widget );
        }
    }
}
```

PtCompound

This class is used for creating widgets whose functionality (in part) is derived from exported subordinate widgets.

Class hierarchy

PtWidget → PtBasic → PtContainer → PtCompound

Class extensions

```c
typedef struct Pt_compound_class {
    PtContainerClass_t container;
    ushort_t num_subordinates;
    ushort_t num_blocked_resources;
    ushort_t *subordinates;
    ulong_t *blocked_resources;
};
```
PtCompound

PtCompoundClass_t;

#define Pt_SET_NUM_SUBORDINATES
   (Pt_ARG_IS_NUMBER(PtCompoundClass_t,num_subordinates))

#define Pt_SET_SUBORDINATES
   (Pt_ARG_IS_POINTER(PtCompoundClass_t,subordinates))

#define Pt_SET_NUM_BLOCKED_RESOURCES
   (Pt_ARG_IS_NUMBER(PtCompoundClass_t,num_blocked_resources))

#define Pt_SET_BLOCKED_RESOURCES
   (Pt_ARG_IS_POINTER(PtCompoundClass_t,blocked_resources))

Pt_SET_NUM_SUBORDINATES
   Indicates the number of subordinate widgets to be exported.

Pt_SET_SUBORDINATES
   An array of offsets to widget pointers to subordinate widgets. These widgets
   must be created in the compound widget’s Defaults method.

Pt_SET_NUM_BLOCKED_RESOURCES
   Indicates the number of resources to block.

Pt_SET_BLOCKED_RESOURCES
   An array of resources to block.

Methods

PtCompound defines the class methods described below.

Defaults method

Inherits all defaults from PtContainer.

Initialization method

None.

Extent method

Inherited from PtContainer.

Connection method

None.

Realization method

None.
PtGenList

This class is used for creating list widgets.

Class hierarchy

PtWidget → PtBasic → PtContainer → PtCompound → PtGenList

Class extensions

typedef void PtGenListDrawF_t(
    PtWidget_t *widget,
    PtGenListItem_t *item, unsigned index,
    unsigned nitems,
    PhRect_t *where /* Modify *where if needed */);

typedef int PtGenListMouseF_t(
    PtWidget_t *wgt,
PtGenListItem_t *item,
unsigned index,
PhPoint_t *where, /* relative to the item, can modify */
int column,
PhEvent_t *ev
);

typedef int PtGenListKeyF_t(
    PtWidget_t *wgt,
    PhEvent_t *ev, PhKeyEvent_t *kev,
    PtGenListItem_t *newcur, unsigned newpos
);

typedef void PtGenListSelectF_t(
    PtWidget_t *wgt, PtGenListItem_t *item, int pos, int column,
    int nitems, int subtype, PhEvent_t *ev
);

typedef PtWidget_t *PtGenListInflateF_t(
    PtWidget_t *widget, PtWidget_t *parent,
    PtGenListItem_t *item, unsigned index,
    int column, PhArea_t *area
);

typedef void PtGenListDrawBackgroundF_t(
    PtWidget_t *widget,
    PhRect_t const *canvas, PhRect_t const *empty
);

typedef int PtGenListDndCallbackF_t(
    PtWidget_t *widget,
    PtGenListItem_t const *item, int index, PhEvent_t *event,
    unsigned long *pflags, unsigned int *action
);

typedef const PtGenListItemAttrs_t *PtGenListGetItemAttrsF_t(
    PtWidget_t *widget, PtGenListItem_t const *item
);

typedef struct Pt_gen_list_widget_class {
    PtCompoundClass_t compound;
    PtGenListDrawF_t *list_draw_f;
    PtGenListMouseF_t *list_mouse_f;
    PtGenListKeyF_t *list_key_f;
    PtGenListSelectF_t *list_select_f;
    PtGenListInflateF_t *list_inflate_f;
    PtGenListDrawBackgroundF_t *list_draw_background_f;
    PtGenListDndCallbackF_t *list_dnd_callback_f;
    PtGenListGetItemAttrsF_t *list_itemattrs_f;
} PtGenListClass_t;

#define Pt_SET_LIST_DRAW_F
    Pt_ARG_IS_POINTER( PtGenListClass_t, list_draw_f )
#define Pt_SET_LIST_MOUSE_F
    Pt_ARG_IS_POINTER( PtGenListClass_t, list_mouse_f )
#define Pt_SET_LIST_KEY_F
    Pt_ARG_IS_POINTER( PtGenListClass_t, list_key_f )
#define Pt_SET_LIST_SELECT_F
    Pt_ARG_IS_POINTER( PtGenListClass_t, list_select_f )
#define Pt_SET_LIST_INFLATE_F
    Pt_ARG_IS_POINTER( PtGenListClass_t, list_inflate_f )
#define Pt_SET_LIST_DRAW_BACKGROUND_F

Methods

PtGenList defines the class methods described below.

Defaults method

```c
widget->flags |= Pt_HIGHLIGHTED | Pt_ETCH_HIGHLIGHT |
               Pt_SET | Pt_GETS_FOCUS | Pt_FOCUS_RENDER;
widget->eflags &= ~Pt_DAMAGE_ON_FOCUS;
widget->resize_flags &= ~Pt_RESIZE_XY_BITS;
widget->border_width = 1; //2;
list->columns = & list->dfltcol;
list->ncolumns = 1;
list->flags = Pt_LIST_SCROLLBAR_AUTORESIZE |
              Pt_LIST_VSCROLLBAR_AS_REQUIRED |
              Pt_LIST_BALLOONS_IN_COLUMNS;
set_selmode( list, list->sel_mode = Pt_BROWSE_MODE );
list->scroll_rate = 2;
list->selection_fill_color = PgRGB( 142, 162, 155 );
list->selection_text_color = Pg_WHITE;
list->balloon_bg = Pg_BALLOONCOLOR;
list->balloon_fg = Pg_BLACK;
list->font = Pt_GET_MAGIC_FONT_NAME( TextFont09 );
list->dnd_selection_color = PgRGB( 216, 216, 216 );
getfontsize( list );

basic->margin_width =
basic->margin_height = 0;
basic->flags = Pt_ALL_ETCHES | Pt_OPAQUE_ETCHES |
               Pt_ALL_OUTLINES /*| Pt_LEFT_INLINE*/ |
               Pt_FLAT_FILL;
basic->inline_color = Pg_GREY;
basic->fill_color = PgGrey( 244 );
widget->eflags &= ~Pt_DAMAGE_ON_FOCUS;
list->slider_width = SCROLLBAR_DEFAULT_WIDTH;
list->max_top = list->top_item = 1;
```

Initialization method

None.

Extent method

If a PtDivider widget is the child, the Extent method:

- calls the Extent method of PtDivider via PtExtentWidget()
- adjusts the size of the PtDivider widget according to the resize policy and the Pt_LIST_SNAP flag

Then the Extent method of PtCompound is invoked via PtSuperClassExtent().
Connection method

None.

Realization method

Realizes the scrollbar if necessary and registers a balloon callback with the parent.

Draw method

Renders the widget’s border, margins, and (possibly) background. Then the List Draw method is called to draw the list.

Unrealization method

Deregisters a balloon callback with the parent.

Destruction method

None.

Got Focus method

If the Pt_FOCUS_RENDER flag is set, the Got Focus method damages the current item (see “Current item” in the description of PtGenList in the Widget Reference). If the Pt_SELECTION_MODE_AUTO flag is set but the Pt_SELECTION_MODE_NOFOCUS flag isn’t set and no items are selected, the current item is selected. Then the Got Focus method of PtCompound is invoked via PtSuperClassGotFocus().

Lost Focus method

If necessary, the current item is damaged. Then the Lost Focus method of PtCompound is invoked via PtSuperClassLostFocus().

Calc Opaque Rect method

Inherited from PtBasic.

Child Created method

Sets the child redirector function to PtCompoundRedirect() via the Pt_SET_CHILD_REDIRECT_F manifest defined by PtContainer. This prevents the list widget from having more than one PtDivider child.

Child Realized method

If the child isn’t the scrollbar (it’s a divider), the Child Realized method:

- Sets list->divider to the widget pointer of the child.
- Adjusts the top margin.
- Sets the orientation of the child to horizontal.
- Sets the child’s Pt_ARG_DIVIDER_OFFSET resource and the Pt CALLBACKS_ACTIVE flag.
PtGenList

- Copies the Pt_ARG_DIVIDER_SIZES resource of the child to the Pt_ARG_LIST_COLUMN_POS resource of the list widget.
- Attaches a function to the Pt_CB_DIVIDER_DRAG callback list.

Child Moved/Resized method

If the child is a divider (i.e. list->divider), the Child Moved/Resized method sets the divider’s Pt_ARG_DIVIDER_OFFSET resource and adjusts the top margin of the list widget.

Child Unrealized method

If the child is the same as list->divider, the Child Unrealized method adjusts the top margin, removes the callback, and sets list->divider to NULL.

Child Destroyed method

Sets the child redirector function to a private function that accepts a PtDivider child only.

Child Setting Resources method

Inherited from PtContainer.

Child Getting Resources method

Inherited from PtContainer.

Child Getting Focus method

Inherited from PtContainer.

Child Losing Focus method

Inherited from PtContainer.

List Draw method

None.

List Mouse method

None.

List Key method

None.

List Select method

None.
List Inflate method

None.

List Attributes method

None.

Widget actions

PtGenList is sensitive to the following events:

- Ph_EV_KEY
- Ph_EV_BUT_PRESS
- Ph_EV_PTR_MOTION_BUTTON
- Ph_EV_BUT_RELEASE
- Ph_EV_BUT_REPEAT

The callback function invokes the List Key or List Mouse method and performs the selection.

If your widget class defines Pt_SET_RAW_CALLBACKS, keep this in mind: if your raw callback is sensitive to one of the events listed above and the callback returns Pt_END or Pt_HALT, PtGenList behavior for that event won’t occur unless you invoke PtGenList’s raw handlers within the callback prior to returning. This can be done via PtSuperClassRawEvent().

Resource definitions

```c
static const PtResourceRec_t resources[] =
{
    {Pt_ARG_AREA, gl_arg_resize, NULL,
     Pt_ARG_IS_STRUCT( PtWidget_t, area )},
    {Pt_ARG_DIM, gl_arg_resize, NULL,
     Pt_ARG_IS_STRUCT( PtWidget_t, area.size )},
    {Pt_ARG_FLAGS, arg_wflags, NULL,
     Pt_ARG_IS_FLAGS( PtWidget_t, flags )},
    {Pt_ARG_POS, gl_arg_resize, NULL,
     Pt_ARG_IS_STRUCT( PtWidget_t, area.pos )},
    {Pt_ARG_COLOR, arg_txtcolor, NULL,
     Pt_ARG_IS_COLOR( PtBasicWidget_t, color )},
    {Pt_ARG_MARGIN_HEIGHT, gl_arg_resize, NULL,
     Pt_ARG_IS_NUMBER( PtGenListWidget_t, marg_height )},
    {Pt_ARG_MARGIN_WIDTH, gl_arg_resize, NULL,
     Pt_ARG_IS_NUMBER( PtGenListWidget_t, marg_width )},
    {Pt_ARG_LIST_FLAGS, gl_arg_flags, NULL},
};
```
Functions

static void arg_resize()

This function:

- adjusts the number of displayed items
- resizes the scrollbar
- adjusts the Pt_ARG_DIVIDER_OFFSET resource of the widget pointed to by list->divider

static void arg_font()

This function:

- allocates a copy of the new font name and frees the old string
- stores the height of the new font in list->font_height

static void arg_top_pos()

This function:
PtGenTree

This class is used for creating tree widgets.

Class hierarchy

PtWidget → PtBasic → PtContainer → PtCompound → PtGenList → PtGenTree

- calculates a new range of displayed items
- blits some of the currently visible items to their new positions (blits only the items that will remain visible after the change)
- adjusts the scrollbar position
- invokes the Pt_CB_SCROLL_MOVE callback

static void arg_selmode()

This function:
- clears the current selection
- if the new value is valid, sets list->sel_mode, list->sel_xmode, and list->sel_flags

static void arg_flags()

For Pt_ARG_LIST_FLAGS, this function:
- adjusts the Pt_GETS_FOCUS flag of the scrollbar if the Pt_LIST_SCROLLBAR_GETS_FOCUS flag is changed
- adjusts list->sel_xmode and list->sel_flags if the Pt_LIST_NON_SELECT flag is changed

static void arg_setsb()

This function:
- checks if allowed resources were specified
- calls PtSetResources() for the scrollbar

static void arg_getsb()

Calls PtGetResources() for the scrollbar.

static void arg_columns()

This function:
- saves the new value
- damages the appropriate portions of the widget

For more information, see the chapter on Creating a List Widget.
### Class extensions

```c
typedef void PtGenTreeDrawItemF_t(  
    PtWidget_t *widget, PtGenTreeItem_t *item,  
    PhRect_t const *where, int lmargin, int rmargin  
);

typedef int PtGenTreeItemStateF_t(  
    PtWidget_t *widget, PtGenTreeItem_t *item,  
    PhEvent_t *event, int reason  
);

typedef struct Pt_gen_tree_widget_class {  
    PtGenListClass_t list;  
    PtGenTreeDrawItemF_t *tree_draw_item_f;  
    PtGenTreeItemStateF_t *tree_state_f;  
} PtGenTreeClass_t;
```

```c
#define Pt_SET_TREE_DRAW_F  
   Pt_ARG_IS_POINTER( PtGenTreeClass_t, tree_draw_item_f )
#define Pt_SET_TREE_STATE_F  
   Pt_ARG_IS_POINTER( PtGenTreeClass_t, tree_state_f )
```

### Methods

**PtGenTree** defines the class methods described below.

#### Defaults method

```c
tree->flags = Pt_TREE_HAS_BUTTONS |  
    Pt_TREE_TO_LEFT | Pt_TREE_TO_RIGHT |  
    Pt_TREE_INDENT_BUTTONS |  
    Pt_TREE_SHOW_CONNECTORS;  

tree->list.gflags = Pt_GEN_LIST_SHOW_DAMAGED |  
    Pt_GEN_LIST_ITEM_BACKGROUND;  

tree->list.selection_fill_color = PgRGB( 142, 162, 155 );  

tree->line_spacing = 3;  

tree->line_color = PgRGB( 239, 239, 239 );  

```

#### Initialization method

None.

#### Extent method

Inherited from **PtGenList**.

#### Connection method

None.

#### Realization method

None.
Draw method

Inherited from PtGenList.

Unrealization method

None.

Destruction method

None.

Got Focus method

Inherited from PtGenList.

Lost Focus method

Inherited from PtGenList.

Calc Opaque Rect method

Inherited from PtBasic.

Child Created method

Inherited from PtGenList.

Child Realized method

Inherited from PtGenList.

Child Moved/Resized method

Inherited from PtGenList.

Child Unrealized method

Inherited from PtGenList.

Child Destroyed method

Inherited from PtGenList.

Child Setting Resources method

Inherited from PtContainer.

Child Getting Resources method

Inherited from PtContainer.

Child Getting Focus method

Inherited from PtContainer.
Child Losing Focus method

Inherited from PtContainer.

List Draw method

For each visible item that has the Pt_LIST_ITEM_DAMAGED flag set, the List Draw method calls the Tree Draw Item method and draws the tree ornaments.

List Mouse method

Invokes the Pt_CB_GEN_TREE_INPUT callback list. Depending on the result of the callback and the pointer position, the List Mouse method returns Pt_CONTINUE, returns Pt_END, or calls PtGenTreeCollapse() or PtGenTreeExpand() (see the Photon Widget Reference) and returns Pt_END.

List Key method

Invokes the Pt_CB_GEN_TREE_INPUT callback list. Depending on the result of the callback and the key code, the List Key method returns Pt_CONTINUE, returns Pt_END, or calls PtGenTreeCollapse() or PtGenTreeExpand() (see the Photon Widget Reference) and returns Pt_HALT.

Tree Draw Item method

None.

Tree Item State method

None.

Widget actions

None.

Resource definitions

```c
static const PtResourceRec_t resources[] = {
    { Pt_ARG_TREE_FLAGS, arg_flags, NULL,
        Pt_ARG_IS_FLAGS( PtGenTreeWidget_t, flags ) },
    { Pt_ARG_TREE_LINE_SPACING, arg_line_spacing, NULL,
        Pt_ARG_IS_NUMBER( PtGenTreeWidget_t, line_spacing ) },
    { Pt_ARG_TREE_LINE_COLOR, Pt_CHANGE_RESIZE_REDRAW, NULL,
        Pt_ARG_IS_NUMBER( PtGenTreeWidget_t, line_color ) },
    { Pt_ARG_TREE_MARGIN_COLOR, Pt_CHANGE_RESIZE_REDRAW, NULL,
        Pt_ARG_IS_NUMBER( PtGenTreeWidget_t, margin_color ) },
    { Pt_CB_GEN_TREE_INPUT, Pt_CHANGE_INVISIBLE, NULL,
        Pt_ARG_IS_CALLBACK( PtGenTreeWidget_t, input_cb ) }
};
```
Functions

static void arg_flags()

This function:

- Adjusts list->gflags if the Pt_TREE_TO_RIGHT flag has changed.
- Damages the widget or marks it for resizing.

For more information, see the chapter on Creating a Tree Widget.

PtLabel

The PtLabel class provides fundamental multiline-text string and image handling.

Class hierarchy

PtWidget → PtBasic → PtLabel

Methods

PtLabel defines the class methods described below.

Defaults method

widget->flags |= Pt_FOCUS_RENDER;
widget->resize_flags |= Pt_RESIZE_XY_AS_REQUIRED;
label->basic.flags = Pt_ALLETCHES | Pt_ALLBEVELS |
        Pt_FLAT_FILL;
label->basic.fill_color = Pg_TRANSPARENT;
label->font = strdup("TextFont09");
label->flags |= Pt_LABEL_SELECT_SHIFT;
label->uline1 = Pg_BLACK;
label->uline2 = Pg_TRANSPARENT;
label->uline_type = (ushort_t)Pt_NO_ULINE;
label->type = Pt_Z_STRING;
label->text_image_spacing = 2;
label->h_alignment = Pt_LEFT;
label->v_alignment = Pt_CENTER;
label->string = strdup("*");
label->inflate_f = PtInflateBalloon;
label->balloon_fill_color = Pg_BALLOONCOLOR;
label->balloon_color = Pg_BLACK;
label->balloon_text = strdup("*");
if ( _Ph_->draw_context->gin.cmd_buf_limit < 4096 )
    PgSetDrawBufferSize(4096);

Initialization method

None.

Extent method

Determines the required canvas size based on the widget’s text string or image data in conjunction with label->type (Pt_Z_STRING, Pt_TEXT_IMAGE ), margins, etc. The Extent method also takes into account multiple lines of text and underlining.
Connection method

If label->flags has the Pt_SHOW_BALLOON flag set, a balloon callback is attached to the parent widget.

Realization method

Inherited from PtBasic.

Draw method

Calls the Draw method of PtBasic via PtSuperClassDraw(). Draws the text in the label using the specified font, color, etc.

Unrealization method

If this label has a balloon displayed (label->balloon_widget != NULL), that widget is destroyed. If label->flags has the Pt_SHOW_BALLOON flag set, it detaches the balloon callback from the parent.

Got focus method

Inherited from PtBasic.

Lost Focus method

Inherited from PtBasic.

Calc Opaque Rect method

If label->type is Pt_Z_STRING, the Calc Opaque Rect method is called via PtSuperClassCalcOpaque().

If label->type is Pt_IMAGE, label->flags has its Pt_OPAQUE flag set.

If label->type is Pt_BITMAP and label->data->image->type is Pg_BITMAP_BACKFILL, the Pt_OPAQUE bit of widget->flags is set.

Widget actions

None.

Resource definitions

static const PtResourceRec_t resources[] =
{
  { Pt_ARG_BALLOON_TEXT, (void*)PtModifyBallonText, NULL,
    Pt_ARG_IS_STRING( PtLabelWidget_t, balloon_text ) },
  { Pt_ARG_FLAGS, set_flags, NULL,
    Pt_ARG_IS_FLAGS( PtWidget_t, flags ) },
  { Pt_ARG_HORIZONTAL_ALIGNMENT, Pt_CHANGE_REDRAW, NULL,
    Pt_ARG_IS_NUMBER( PtLabelWidget_t, h_alignment ) },
  { Pt_ARG_LABEL_IMAGE, Pt_CHANGE_CANVAS_REDRAW, NULL,
    Pt_ARG_IS_IMAGE( PtLabelWidget_t, data ) },
  { Pt_ARG_LABEL_FLAGS, label_modify_flags, NULL,
    Pt_ARG_LABEL_FLAGS( PtLabelWidget_t, flags ) },
  { Pt_ARG_LABEL_TYPE, Pt_CHANGE_RESIZE_REDRAW, NULL,
    Pt_ARG_IS_NUMBER( PtLabelWidget_t, type ) },
Functions

```c
static int label_modify_flags( PtWidget_t *widget, 
    PtArg_t const *argt );
```

If the Pt_SHOW_BALLOON flag is being set, the balloon callback is attached to the parent. If the Pt_SHOW_BALLOON flag is being cleared, the balloon callback is detached from the parent. The value of `label->flags` is based on `argt->value` and `argt->len`. This is typically done as follows:

```c
label->flags = (label->flags & ~argt->len) | argt->value;
```

```c
int PtModifyLabelString( PtWidget_t *widget, 
    PtArg_t const *argt );
```

This frees the current string, allocates enough space for the new string, copies in the new string, flags the widget to be resized (widget->flags |= Pt_WIDGET_RESIZE;), and damages the widget.

If a balloon is displayed and Pt_ARG_BALLOON_TEXT is not set, the balloon is destroyed and a new balloon is inflated with the new string. If `label->flags` has the
Pt_BALLOON_AS_REQUIRED bit set, the new balloon is created only if the label is clipped by its parent.

```c
static int PtModifyBalloonText( PtWidget_t *widget,
                                 PtArg_t const *arg );
```

This frees the current balloon text, allocates enough space for the new balloon text and copies it in. If a balloon is currently displayed, the balloon is destroyed and a new balloon is inflated with the new text. If label->flags has the Pt_BALLOON_AS_REQUIRED bit set, the new balloon is created only if the label is clipped by its parent.

**PtGraphic**

This class provides standard graphic resources such as Pt_ARG_POINTS and Pt_ARG_ORIGIN.

**Class hierarchy**

PtWidget → PtBasic → PtGraphic

**Methods**

PtGraphic defines the class methods described below.

**Defaults method**

```c
graphic->line_cap = Pg_BUTT_CAP;
graphic->line_join = Pg_MITER_JOIN;
graphic->basic.margin_height = 0;
graphic->basic.margin_width = 0;
graphic->basic.fill_color = Pg_TRANSPARENT;
```

**Initialization method**

None.

**Extent method**

Determines the render rectangle for the graphic widget, given the area, point array, and current settings of graphic flags.

**Connection method**

None.

**Realization method**

None.
Draw method

Inherited from \texttt{PtBasic}.

Unrealization method

None.

Destruction method

None.

Got Focus method

Inherited from \texttt{PtBasic}.

Lost Focus method

Inherited from \texttt{PtBasic}.

Calc Opaque Rect method

Inherited from \texttt{PtBasic}.

Widget actions

None.

Resource definitions

\begin{verbatim}
static const PtResourceRec_t resources[] = {
    Pt_ARG_AREA, graphic_modify_area, 0,
    Pt_ARG_IS_STRUCT( PtWidget_t, area ), 0,
    Pt_ARG_DIM, graphic_modify_area, 0,
    Pt_ARG_IS_STRUCT( PtWidget_t, area.size ) |
    Pt_ARG_PARM( sizeof( PhPoint_t ) ), 0,
    Pt_ARG_DASH_LIST, Pt_CHANGE_REDRAW, 0,
    Pt_ARG_IS_ARRAY( PtGraphicWidget_t, dash_list ),
    Pt_ARG_IS_NUMBER( PtGraphicWidget_t, dash_len ),
    Pt_ARG_GRAPHIC_FLAGS, Pt_CHANGE_RESIZE, 0,
    Pt_ARG_IS_FLAGS( PtGraphicWidget_t, flags ),0,
    Pt_ARG_LINE_WIDTH, Pt_CHANGE_RESIZE_REDRAW, 0,
    Pt_ARG_LINE_WIDTH, Pt_CHANGE_RESIZE_REDRAW, 0,
    Pt_ARG_LINE_JOIN, Pt_CHANGE_REDRAW, 0,
    Pt_ARG_LINE_JOIN, Pt_CHANGE_REDRAW, 0,
    Pt_ARG_LINE_CAP, Pt_CHANGE_REDRAW, 0,
    Pt_ARG_LINE_CAP, Pt_CHANGE_REDRAW, 0,
    Pt_ARG_ORIGIN, Pt_CHANGE_RESIZE_REDRAW, 0,
    Pt_ARG_ORIGIN, Pt_CHANGE_RESIZE_REDRAW, 0,
    Pt_ARG_IS_STRUCT( PtGraphicWidget_t, origin ),0,
    Pt_ARG_POINTS, Pt_CHANGE_RESIZE_REDRAW, 0,
    Pt_ARG_ARRAY( PtGraphicWidget_t, point_array),
    Pt_ARG_IS_NUMBER( PtGraphicWidget_t, npoints),
    Pt_CB_RESCALE, Pt_CHANGE_INVISIBLE, 0,
    Pt_ARG_IS_CALLBACK_LIST( PtGraphicWidget_t, rescale ), 0,
    Pt_ARG_DASH_SCALE, Pt_CHANGE_REDRAW, 0,
    Pt_ARG_DASH_SCALE, Pt_CHANGE_REDRAW, 0,
    Pt_ARG_IS_NUMBER( PtGraphicWidget_t, dash_scale ), 0,
};
\end{verbatim}
Functions

`graphic_modify_area()`

Updates the widget’s area or dimension and invokes the `Pt_CB_RESCALE` callback.

**PtGauge**

This class provides standard gauge resources such as maximum, minimum, and orientation.

Class hierarchy

`PtWidget` → `PtBasic` → `PtGauge`

Methods

`PtGauge` defines the class methods described below.

Defaults method

```c
gauge->font = strdup("TextFont09");
gauge->flags = Pt_GAUGE_HORIZONTAL;
gauge->maximum = 100;
```

Initialization method

None.

Extent method

Inherited from `PtBasic`.

Connection method

None.

Realization method

None.

Draw method

Inherited from `PtBasic`.

Unrealization method

None.

Destruction method

None.
Got Focus method

Inherited from PtBasic.

Lost Focus method

Inherited from PtBasic.

Calc Opaque Rect method

Inherited from PtBasic.

Widget actions

None.

Resource definitions

```c
static const PtResourceRec_t resources[] =
{
    { Pt_ARG_GAUGE_FLAGS, Pt_CHANGE_INVISIBLE, NULL,
        Pt_ARG_IS_FLAGS( PtGaugeWidget_t, flags ),
        Pt_GAUGE_RO_FLAGS },
    { Pt_ARG_GAUGE_FONT, Pt_CHANGE_CANVAS_REDRAW, NULL,
        Pt_ARG_IS_STRING( PtGaugeWidget_t, font ) },
    { Pt_ARG_GAUGE_MINIMUM, Pt_CHANGE_CANVAS_REDRAW, NULL,
        Pt_ARG_IS_NUMBER( PtGaugeWidget_t, minimum ) },
    { Pt_ARG_GAUGE_MAXIMUM, Pt_CHANGE_CANVAS_REDRAW, NULL,
        Pt_ARG_IS_NUMBER( PtGaugeWidget_t, maximum ) },
    { Pt_ARG_GAUGE_VALUE, Pt_CHANGE_REDRAW, NULL,
        Pt_ARG_IS_NUMBER( PtGaugeWidget_t, value ) },
    { Pt_ARG_ORIENTATION, Pt_CHANGE_RESIZE_REDRAW, NULL,
        Pt_ARG_IS_BOOLEAN( PtGaugeWidget_t, flags ),
        Pt_GAUGE_HORIZONTAL },
    { Pt_ARG_GAUGE_VALUE_PREFIX, Pt_CHANGE_CANVAS_REDRAW, NULL,
        Pt_ARG_IS_STRING( PtGaugeWidget_t, prefix ) },
    { Pt_ARG_GAUGE_VALUE_SUFFIX, Pt_CHANGE_CANVAS_REDRAW, NULL,
        Pt_ARG_IS_STRING( PtGaugeWidget_t, suffix ) },
    { Pt_ARG_GAUGE_H_ALIGN, Pt_CHANGE_CANVAS, NULL,
        Pt_ARG_IS_NUMBER( PtGaugeWidget_t, h_alignment ) },
    { Pt_ARG_GAUGE_V_ALIGN, Pt_CHANGE_CANVAS, NULL,
        Pt_ARG_IS_NUMBER( PtGaugeWidget_t, v_alignment ) },
};
```

Functions

None.
Chapter 5
Creating a List Widget

In this chapter...

Overview 109
Item structure 109
Widget instance structure 110
Methods 112
Convenience functions 115
PtSuperClassGenListDraw() 118
PtSuperClassGenListInflate() 119
PtSuperClassGenListKey() 121
PtSuperClassGenListMouse() 123
PtSuperClassGenListSelect() 125
Overview

The Photon library provides the generic list widget class PtGenList for creating custom list widgets. The PtGenList class displays a vertical list of any number of items. A user can select one or more items from this list, depending on the selection policy.

The child class shouldn’t override the Draw method or consume mouse or key events. Instead, it should define a List Draw method for drawing and may define a Mouse and/or Key Event method if additional handling of those event types is required. Usually, the child class will also define a Selection method that will be called by the PtGenList event handler.

Item structure

The PtGenList class never allocates or frees items. This is the responsibility of the child class. The PtGenList class expects each item to be represented by a PtGenListItem_t structure — see the Photon Widget Reference.

When you call a PtGenList*() convenience function, you give it a pointer to the PtGenListItem_t structure as an item. When PtGenList calls one of your methods, it provides a pointer to the PtGenListItem_t structure as an item.

Since a child class will most likely need additional data describing the item, it will usually define its item as a structure whose first member is PtGenListItem_t. This makes conversions to and from PtGenListItem_t easy.

The child class is responsible for allocating items and adding them to the widget. Before adding an item to the widget, the members of the item structure should be set to the required values. After adding the item to the widget, don’t modify the PtGenListItem_t directly — use the convenience functions instead.

The PtGenListItem_t structure is defined as follows:

```c
typedef struct Pt_genlist_item {
    unsigned flags;
    PhDim_t size;
    PtGenListItem_t *next, *prev;
} PtGenListItem_t;
```

flags

The following flags describe the state of the item:

**Pt_LIST_ITEM_SELECTED**

This item is selected. Use PtGenListSelect() and PtGenListUnselect() to modify this flag.

**Pt_LIST_ITEM_CURRENT**

This item is the current item (see “Current item” in the description of PtGenList in the Widget Reference). Use PtGenListGoto() to modify this flag.
Widget instance structure

Some members of the \texttt{PtGenListWidget\_t} structure may be used by the child class. In general, don’t modify the structure members directly; use the provided resources and convenience functions instead.

For information on the correspondence between structure members and resources, see the “Resource definitions” section for \texttt{PtGenList} in the chapter on Using Widget Superclasses.

Flags that affect the behavior of the widget. The child class will probably set all or most of these flags to a fixed value; however, if a class has to modify \texttt{gflags} after the widget has been created, use \texttt{PtGenListSetGflags()}

\begin{itemize}
  \item \texttt{Pt\_GEN\_LIST\_NO\_BACKGROUND}
    
    If set, the Draw method won’t draw the background (under the items) or margins.
\end{itemize}
Pt_GEN_LIST_ITEMBACKGROUND

If this flag is set but the Pt_GEN_LIST_NO_BACKGROUND flag is clear, the Draw method will draw the background beneath each item and call the List Draw method once for each item that needs to be drawn.

Pt_GEN_LIST_NO_CLIPPING

If this flag is clear, the Draw method will set clipping to the widget’s canvas before calling the List Draw method. The clipping won’t be set if no items are wider than the canvas.

Pt_GEN_LIST_SHOW_DAMAGED

If this flag is set, the Draw method will scan the damage list and set the Pt_LIST_ITEM_DAMAGE flag in the items that need to be drawn. The List Draw method won’t be called at all if no items are damaged.

Pt_GEN_LIST_FULL_WIDTH

If this flag is set, the space to the right of an item is considered part of the item — the width stored in the item is treated as a suggested minimum. If that space is damaged, the item will be marked as damaged. If this flag isn’t set, the space to the right of an item is treated like the margin. Mouse clicks on that space are ignored.

Pt_GEN_LIST_NO_AUTOFOCUS

If this flag is clear, giving focus to the widget when there’s no current item in that widget will automatically set the current item to the first displayed item (see “Current item” in the description of PtGenList in the Widget Reference).

Pt_LIST_Balloons_IN_COLUMNS

If this flag is set and the widget has columns, the widget will destroy and recreate the balloon when the mouse pointer crosses column boundaries.

sel_xmode and sel_xflags

The “decomposed” value of the current selection mode. The value of sel_xmode can be one of:

- Pt_SELECTION_MODE( Pt_SELECTION_MODE_NONE )
- Pt_SELECTION_MODE( Pt_SELECTION_MODE_SINGLE )
- Pt_SELECTION_MODE( Pt_SELECTION_MODE_MULTIPLE )
- Pt_SELECTION_MODE( Pt_SELECTION_MODE_RANGE )

The sel_xflags member contains the “flags” part of the current selection mode:

- Pt_SELECTION_FLAGS( Pt_SELECTION_MODE_NOSCROLL )
• Pt_SELECTION_FLAGS( Pt_SELECTION_MODE_NOMOVE )
• Pt_SELECTION_FLAGS( Pt_SELECTION_MODE_NOREST )
• Pt_SELECTION_FLAGS( Pt_SELECTION_MODE_AUTO )
• Pt_SELECTION_FLAGS( Pt_SELECTION_MODE_NOCLEAR )
• Pt_SELECTION_FLAGS( Pt_SELECTION_MODE_TOGGLE )
• Pt_SELECTION_FLAGS( Pt_SELECTION_MODE_NOFOCUS )

`current_item` The pointer to the current item (see “Current item” in the description of PtGenList in the Widget Reference).

`cur_ndx` The index of the current item (the first item on the list has an index of 1).

**Methods**

The methods described in this section are defined in the PtGenList class.

**List Draw method**

The Draw method calls the List Draw method with the following arguments:

```c
PtGenListWidget_t *widget
    The widget pointer.
PtGenListItem_t *items
    A pointer to the PtGenListItem_t structure for the first item that needs to be redrawn.
unsigned index
    The index of the item to be redrawn (the first item on the list has an index of 1).
unsigned nitems
    The number of items the function should look at.
PhRect_t *where
    A pointer to a PhRect_t structure (see the Photon Library Reference) that defines the extent of the items (the function is allowed to modify the PhRect_t structure pointed to by where).
```

**List Mouse method**

The list class’s raw callbacks invoke the Mouse List method on every mouse event if the mouse points to an item. The function can return a value other than Pt_CONTINUE to suppress normal mouse handling.

The List Mouse method is called with the following arguments:

```c
PtGenListWidget_t *widget
    The widget pointer.
```
PtGenListItem_t *item
A pointer to the PtGenListItem_t structure for the item under the mouse cursor.

unsigned index
The index of that item (the first item on the list has an index of 1).

PhPoint_t *where
A pointer to a PhPoint_t structure (see the Photon Library Reference) that defines the mouse position, relative to the item. The function is allowed to modify the structure pointed to by where.

int column
The index of the column under the mouse pointer, or -1 if the pointer isn’t on a column or the widget has no columns.

PhEvent_t const *event
A pointer to a PhEvent_t structure (see the Photon Library Reference) that describes the event.

List Key method
The list class’s raw callbacks invoke the List Key method on every key event. This method should return one of the following values:

Pt_CONTINUE The PtGenList class will normally process the data contained in the PhKeyEvent_t structure. Note that the class’s raw callbacks are allowed to modify the contents of that structure.

Pt_END The widget will ignore the event and the class’s raw callbacks will return Pt_CONTINUE immediately.

Pt_HALT The event will be consumed by the widget, but PtGenList itself won’t take any further action. The class’s raw callbacks return Pt_END immediately.

The List Key method is called with the following arguments:

PtGenListWidget_t *widget
The widget pointer.

PhEvent_t const *event
A pointer to a PhEvent_t structure (see the Photon Library Reference) that describes the event.

PhKeyEvent_t *kevent
A pointer to the event data (which the function can modify). See PhKeyEvent_t in the Photon Library Reference.
PtGenListItem_t *newcur

A pointer to the PtGenListItem_t structure for the new current item
(if the event is processed normally).

int index

The index of that item (the first item on the list has an index of 1).

List Select method

The list class’s raw callbacks invoke the List Select method when an item is selected
or unselected. The arguments are:

PtGenListWidget_t *list

The widget pointer.

PtGenListItem_t *item

A pointer to a PtGenListItem_t structure. In
Pt_SELECTION_MODE_RANGE selection mode, it’s a pointer to the
first selected item. In other modes, it’s a pointer to the item that has
been selected or unselected.

int index

The index of that item (the first item on the list has an index of 1).

int column

The index of the column under the mouse pointer, or -1 if the pointer
isn’t on a column or the widget has no columns.

int nitems

The current number of selected items (the same as list->sel_count).

int subtype

The selection subtype.

PhEvent_t const *event

A pointer to a PhEvent_t structure (see the Photon Library
Reference) that describes the event.

List Inflate method

The List Inflate method is called whenever a balloon widget needs to be created. The
List Inflate method should create a balloon and return its widget pointer.

The List Inflate method is called with the following arguments:

PtWidget_t *widget

The list widget.

PtWidget_t *parent

The parent widget for the balloon.

PtGenListItem_t *item

A pointer to the PtGenListItem_t structure for the item
under the mouse pointer.
**List Attributes method**

The List Attributes is called from the widget’s draw functions (\texttt{PtGenListDrawString()} and \texttt{PtGenListDrawBackground()}). It should return a pointer to a \texttt{PtGenListItemAttrs\_t} attribute structure filled in with information about how to draw the item, and has the following arguments:

\begin{verbatim}
PtWidget\_t *widget
    The widget pointer.
PtGenListItem\_t *item
    A pointer to the item that’s being drawn.
\end{verbatim}

If the method returns NULL, the calling draw function uses the widget’s resources. Also, if any of the colors in the attributes structure are set to \texttt{Pg\_TRANSPARENT}, or the font is NULL, the corresponding widget-wide setting is used.

**Convenience functions**

The \texttt{PtGenList} class defines the following convenience functions (see the Photon \textit{Widget Reference}):  

\begin{verbatim}
PtGenListAddItems()
    Add items to a list.
PtGenListAllItems()
    Get pointers to all the items in a list.
PtGenListClearSelection()
    Clear the selection.
PtGenListDamageItem()
    Redraw an item when its data has been changed.
PtGenListDrawBackground()
    Draw the background of a list.
\end{verbatim}
Convenience functions

\begin{itemize}
\item \texttt{PtGenListDrawString()}
  \begin{itemize}
  \item Draw a string.
  \end{itemize}
\item \texttt{PtGenListFirstItem()}
  \begin{itemize}
  \item Return a pointer to the first item in a list.
  \end{itemize}
\item \texttt{PtGenListGetCurrent()}
  \begin{itemize}
  \item Return a pointer to the current item in a list.
  \end{itemize}
\item \texttt{PtGenListGetSelIndexes()}
  \begin{itemize}
  \item Get the indexes of the selected items.
  \end{itemize}
\item \texttt{PtGenListGoto()}
  \begin{itemize}
  \item Set the current item so that the new current item is visible.
  \end{itemize}
\item \texttt{PtGenListHold()}
  \begin{itemize}
  \item Prevent visible repair of a list widget.
  \end{itemize}
\item \texttt{PtGenListItemIndex()}
  \begin{itemize}
  \item Find the index of an item.
  \end{itemize}
\item \texttt{PtGenListItemRealloc()}
  \begin{itemize}
  \item Reallocate memory for an item.
  \end{itemize}
\item \texttt{PtGenListLastItem()}
  \begin{itemize}
  \item Return a pointer to the last item in a list.
  \end{itemize}
\item \texttt{PtGenListLockItem()}
  \begin{itemize}
  \item Lock an item so it can be resized.
  \end{itemize}
\item \texttt{PtGenListRelease()}
  \begin{itemize}
  \item Release a hold on visible repairs of a list widget.
  \end{itemize}
\item \texttt{PtGenListRemoveItems()}
  \begin{itemize}
  \item Remove items from a list.
  \end{itemize}
\item \texttt{PtGenListResize()}
  \begin{itemize}
  \item Resize a list widget.
  \end{itemize}
\item \texttt{PtGenListSelect()}
  \begin{itemize}
  \item Select an item in a list.
  \end{itemize}
\item \texttt{PtGenListSelectedItems()}
  \begin{itemize}
  \item Get pointers to the selected items.
  \end{itemize}
\item \texttt{PtGenListSetGflags()}
  \begin{itemize}
  \item Modify the \textit{gflags} field of the widget.
  \end{itemize}
\item \texttt{PtGenListSetSelIndexes()}
  \begin{itemize}
  \item Set the selection indexes.
  \end{itemize}
\item \texttt{PtGenListShow()}
  \begin{itemize}
  \item Set the current position so a given item is visible.
  \end{itemize}
\item \texttt{PtGenListUnlockItem()}
  \begin{itemize}
  \item Unlock an item so it can be updated.
  \end{itemize}
\end{itemize}
Convenience functions

\[ PtGenListUnselect() \]
Unselect an item in a list.

\[ PtSuperClassGenListDraw() \]
Invoke the Draw List method in a superclass.

\[ PtSuperClassGenListInflate() \]
Invoke the List Inflate method in a superclass.

\[ PtSuperClassGenListKey() \]
Invoke the List Key method in a superclass.

\[ PtSuperClassGenListMouse() \]
Invoke the List Mouse method in a superclass.

\[ PtSuperClassGenListSelect() \]
Invoke the List Select method in a superclass.
Invoke the List Draw method in a superclass

Synopsis:

```c
void PtSuperClassGenListDraw(
    PtWidgetClassRef_t *cref,
    PtWidget_t *widget,
    PtGenListItem_t *item,
    unsigned index,
    unsigned nitems,
    PhRect_t *where );
```

Arguments:

- `cref`: The superclass whose method you want to call.
- `widget`: A pointer to the widget for which to call the superclass’s method.
- `item`: A pointer to the `PtGenListItem_t` structure (see the Widget Reference) for the first item that needs to be redrawn.
- `index`: The index of the item to be redrawn (the first item on the list has an index of 1).
- `nitems`: The number of items the function should look at.
- `where`: A pointer to a `PhRect_t` structure (see the Photon Library Reference) that defines the extent of the items. The function can modify this rectangle.

Description:

This function can be used to invoke the List Draw method of the class defined by `cref`.

Classification:

- Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

- `PtGenList`, `PtGenListItem_t` in the Photon Widget Reference
- `PhRect_t` in the Photon Library Reference
Synopsis:

```c
PtWidget_t *PtSuperClassGenListInflate(
    PtWidgetClassRef_t *cref,
    PtWidget_t *wgt,
    PtWidget_t *parent,
    PtGenListItem_t *item,
    unsigned index,
    int column,
    PhArea_t *area
);
```

Arguments:

- `cref` The superclass whose method you want to call.
- `wgt` A pointer to the widget for which to call the superclass’s method.
- `parent` A pointer to the parent widget for the balloon.
- `item` A pointer to the `PtGenListItem_t` structure (see the Widget Reference) for the first item under the balloon.
- `index` The index of the item (the first item on the list has an index of 1).
- `column` The index of the column under the mouse pointer, or -1 if the pointer isn’t on a column or the widget has no columns.
- `area` A pointer to a `PhArea_t` structure (see the Photon Library Reference) that defines the area (relative to the parent widget) corresponding to the entire item.

Description:

This function can be used to invoke the List Inflate method of the class defined by `cref`.

Returns:

A widget pointer to the balloon instance created by this function.

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
See also:

`PtGenList, PtGenListItem_t` in the Photon Widget Reference

`PhArea_t` in the Photon Library Reference
Synopsis:

```c
int PtSuperClassGenListKey(
    PtWidgetClassRef_t *cref,
    PtWidget_t *wgt,
    PhEvent_t *ev,
    PhKeyEvent_t *kev,
    PtGenListItem_t *newcur,
    unsigned newpos );
```

Arguments:

- **cref**: The superclass whose method you want to call.
- **wgt**: A pointer to the widget for which to call the superclass’s method.
- **ev**: A pointer to a `PhEvent_t` structure (see the Photon Library Reference) that defines the event.
- **kev**: A pointer to a `PhKeyEvent_t` structure (see the Photon Library Reference) that defines the event data. The function can modify this structure.
- **newcur**: A pointer to a `PtGenListItem_t` structure (see the Photon Widget Reference) for the new current item (if the event is processed normally).
- **newpos**: The index of that item (the first item on the list has an index of 1).

Description:

This function can be used to invoke the List Key method of the class defined by `cref`.

Returns:

A nonzero value if the key event was consumed, 0 otherwise.

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
See also:

\texttt{PtGenList}, \texttt{PtGenListItem\_t} in the Photon \textit{Widget Reference}

\texttt{PhEvent\_t}, \texttt{PhKeyEvent\_t} in the Photon \textit{Library Reference}
Invoke the List Mouse method in a superclass

Synopsis:

```c
int PtSuperClassGenListMouse(
   PtWidgetClassRef_t *cref,
   PtWidget_t *wgt,
   PtGenListItem_t *item,
   unsigned index,
   PhPoint_t *where,
   int column,
   PhEvent_t *ev );
```

Arguments:

- `cref` The superclass whose method you want to call.
- `widget` A pointer to the widget for which to call the superclass’s method.
- `item` A pointer to the `PtGenListItem_t` structure (see the Widget Reference) for the item under the mouse pointer.
- `index` The index of that item (the first item on the list has an index of 1).
- `where` A pointer to a `PhPoint_t` structure (see the Photon Library Reference) that defines the mouse position, relative to the item. The function can modify this structure.
- `column` The index of the column under the mouse pointer, or -1 if the pointer isn’t on a column or the widget has no columns.
- `ev` A pointer to a `PhEvent_t` structure (see the Photon Library Reference) that defines the event.

Description:

This function can be used to invoke the List Mouse method of the class defined by `cref`.

Returns:

A nonzero value if the event was consumed, 0 otherwise.

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
See also:

PtGenList, PtGenListItem_t in the Photon Widget Reference
PhEvent_t, PhPoint_t in the Photon Library Reference
Synopsis:

```c
void PtSuperClassGenListSelect(
    PtWidgetClassRef_t *cref,
    PtWidget_t *wgt,
    PtGenListItem_t *item,
    int pos,
    int column,
    int nitems,
    int subtype,
    PhEvent_t *ev );
```

Arguments:

cref  The superclass whose method you want to call.

cwidget A pointer to the widget for which to call the superclass’s method.

citem  A pointer to the PtGenListItem_t structure (see the Widget Reference). In Pt_SELECTION_MODE_RANGE selection mode, it’s a pointer to the first selected item. In other modes, it’s a pointer to the item that has been selected or unselected.

cpos  The index of that item (the first item in the list has an index of 1).

ccolumn  The index of the column under the mouse pointer, or -1 if the pointer isn’t on a column or the widget has no columns.

cnitems  The number of items being selected or unselected.

csubtype  The selection subtype.

cev  A pointer to a PhEvent_t structure (see the Photon Library Reference) that defines the event.

Description:

This function can be used to invoke the List Select method of the class defined by cref.

Classification:

Photon

Safety

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
See also:

- `PtGenList, PtGenListItem_t` in the Photon *Widget Reference*
- `PhEvent_t` in the Photon *Library Reference*
Chapter 6
Creating a Tree Widget

In this chapter...

Overview 129
Item structure 129
Methods 130
Convenience functions 131
PtSuperClassGenTreeDrawItem() 134
PtSuperClassGenTreeItemState() 135
Overview

The **PtGenTree** class is a subclass of **PtGenList**. You can use a **PtGenTree** widget to display a hierarchy of items as lists in a *tree*.

When an item in the hierarchy has an item or list of items below it, that item can be expanded. An expanded item drops a list to reveal all items in the next level below. An expanded item can be collapsed. When an expandable item is collapsed, the list rolls up and its items are concealed. An item can be marked as expandable even if it doesn’t have a list of items below it — you can add the list of items just before the expandable item is expanded.

Any **PtGenTree** widget can contain multiple items at the root level, which contains the first level of items to display. Root-level items have no other items/levels above them — they represent the top of the hierarchy.

A child class of **PtGenTree** isn’t supposed to define a List Draw method. Instead, it should define a Tree Draw Item method that will be called by the List Draw method of **PtGenTree**.

**PtGenTree**’s Draw List method assumes that the **Pt_GEN_LIST_SHOW_DAMAGED** flag is set. The child class should never clear this flag.

The **PtGenTree** class doesn’t use recursive functions. The amount of stack needed by any **PtGenTree** function doesn’t depend on the tree’s size or depth.

Item structure

The item structure used by **PtGenTree** is defined as follows:

```c
typedef struct Pt_gentree_item {
    PtGenListItem_t list;
    struct Pt_gentree_item *father, *son, *brother;
    PhDim_t dim;
} PtGenTreeItem_t;
```

**list** The item structure, of type **PtGenListItem_t** (see the Photon *Widget Reference*) used by the **PtGenList** superclass. The **PtGenTree** class defines some new flags that can be set in `item->list.flags`:

**Pt_TREE_ITEM_EXPANDABLE**

Set this flag to mark an item as expandable. When you add a branch to an item, **PtGenTreeAddFirst()** (see the Photon *Widget Reference*) sets this flag in the parent item automatically.

**Pt_TREE_ITEM_EXPANDED**

The branches of this item are currently on display.
father, son, brother

Tree links. You can use them to traverse the tree structure, but don’t modify them directly — it’s safer to use the convenience functions to modify the tree structure.

dim

A PhDim_t structure (see the Photon Library Reference) that defines the size of the item, excluding the tree ornaments (lines and button).

When an item is added to the widget, the widget calculates item->list.size based on the size specified in item->dim, the minimum height of the item, and the item’s position in the tree. The height of an item in a tree widget is always an even number of pixels — if you specify an odd number for the height in the dim field, a gap of at least one pixel will be displayed between the current item and any other item directly below (see “Current item” in the description of PtGenList in the Widget Reference).

Methods

Tree Draw Item method

The List Draw method calls the Tree Draw Item method with the following arguments:

PtGenTreeWidget_t *widget

The widget pointer.

PtGenTreeItem_t *item

A pointer to the PtGenTreeItem_t structure for the item that needs to be redrawn.

PhRect_t const *where

A pointer to a PhRect_t structure (see the Photon Library Reference) that defines the extent of the item.

int lmargin

If positive, an additional left margin that should be added to the first column.

int rmargin

If positive, an additional right margin that should be added to the last column.

The PtGenTree List Draw method is responsible for drawing the lines and boxes representing the tree structure. The Tree Draw Item method should draw only the item. For example, the Tree Draw Item method of PtTree draws only the image and the string.
Tree Item State method

`PtGenTreeExpand()` and `PtGenTreeCollapse()` (see the Photon Widget Reference) call the Tree Item State method with the following arguments:

```c
PtGenTreeWidget_t *widget
    The widget pointer.
PtGenTreeItem_t *item
    A pointer to the `PtGenTreeItem_t` structure for the item that is being collapsed or expanded.
PhEvent_t *event
    The `event` argument to `PtGenTreeExpand()` or `PtGenTreeCollapse()`. See `PhEvent_t` in the Photon Library Reference.
int reason
    Either `Pt_TREE_EXPANDING` or `Pt_TREE_COLLAPSING`.
```

If `reason` is `Pt_TREE_EXPANDING`, the item is about to be expanded. The Tree Item State method can update the item’s branches before the actual expansion. After the function returns, the widget will display a list of items in the expanded branch.

If an item in the list has its `Pt_TREE_ITEM_EXPANDED` flag set, the items below will be displayed too. The Tree Item State method returns zero to permit the expansion, or a nonzero value to prevent it — `PtGenTreeExpand()` returns this value (see the Photon Widget Reference).

If `reason` is `Pt_TREE_COLLAPSING`, the item has been collapsed. Its branches are concealed and the Tree Item State method can free the associated items. The Tree Item State method returns a nonzero value if the item is destroyed, or zero if the item still exists — `PtGenTreeCollapse()` returns this value (see the Photon Widget Reference).

Convenience functions

The `PtGenTree` class defines the following convenience functions (see the Photon Widget Reference):

```c
PtGenTreeAddAfter()
    Add items after a given item.
PtGenTreeAddFirst()
    Add items in front of any existing items.
PtGenTreeAllItems()
    Get pointers to all the items in the tree.
PtGenTreeClearSelection()
    Clear the selection.
```
Convenience functions

\textit{PtGenTreeCollapse()}  
Collapse a subtree.

\textit{PtGenTreeDamageItem()}  
Redraw an item when its data has changed.

\textit{PtGenTreeExpand()}  
Expand a given subtree.

\textit{PtGenTreeExpandParents()}  
Expand any collapsed ancestors of a given item.

\textit{PtGenTreeFreeAllItems()}  
Free all the items in a tree.

\textit{PtGenTreeFreeItems()}  
Free the items in a subtree.

\textit{PtGenTreeGetCurrent()}  
Get a pointer to the current item.

\textit{PtGenTreeGetSelIndexes()}  
Get the indexes of the selected items.

\textit{PtGenTreeGoto()}  
Set the current item and position so that a given item is visible.

\textit{PtGenTreeItemIndex()}  
Calculate the index of a given item.

\textit{PtGenTreeItemRealloc()}  
Reallocate an item.

\textit{PtGenTreeItemResize()}  
Resize an item.

\textit{PtGenTreeRemoveChildren()}  
Unlink all the children of a given item.

\textit{PtGenTreeRemoveItem()}  
Remove a given item and its children from its parents and siblings.

\textit{PtGenTreeRemoveList()}  
Remove a given items and its siblings from their parent.

\textit{PtGenTreeResize()}  
Resize many items.

\textit{PtGenTreeRootItem()}  
Get a pointer to the first root item.
Convenience functions

\( \text{PtGenTreeSelect()} \)  
Select a given item.

\( \text{PtGenTreeSelectedItems()} \)  
Get pointers to the selected items.

\( \text{PtGenTreeSetSelIndexes()} \)  
Set the selection indexes.

\( \text{PtGenTreeShow()} \)  
Set the current position so that a given item is visible.

\( \text{PtGenTreeUnselect()} \)  
Unselect a given item.

\( \text{PtGenTreeUnselectNonBrothers()} \)  
Unselect all items that aren’t siblings of a given item.

\( \text{PtSuperClassGenTreeDrawItem()} \)  
Invoke the Tree Draw Item method of a given superclass.

\( \text{PtSuperClassGenTreeItemState()} \)  
Invoke the Tree Item State method of a superclass.
PtSuperClassGenTreeDrawItem()

Invoke the Tree Draw Item method of a given superclass

Synopsis:

```c
void PtSuperClassGenTreeDrawItem(
    PtWidgetClassRef_t *cref,
    PtWidget_t *wgt,
    PtGenTreeItem_t *item,
    PhRect_t const *where,
    int lmargin,
    int rmargin );
```

Arguments:

- **cref**  
  The superclass whose method you want to call.

- **widget**  
  A pointer to the widget for which to call the superclass’s method.

- **item**  
  A pointer to a `PtGenTreeItem_t` structure (see the Photon Widget Reference) for the item to be redrawn.

- **where**  
  A pointer to a `PhRect_t` structure (see the Photon Library Reference) that defines the extent of the item.

- **lmargin**  
  If positive, an additional left margin that should be added to the first column.

- **rmargin**  
  If positive, an additional right margin that should be added to the last column.

Description:

This function invokes the Tree Draw Item method of the class defined by `cref`.

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

- `PtGenTree`, `PtGenTreeItem_t` in the Photon Widget Reference
- `PhRect_t` in the Photon Library Reference
PtSuperClassGenTreeItemState()

Invoke the Tree Item State method of a superclass

Synopsis:

```c
int PtSuperClassGenTreeItemState(
    PtWidgetClassRef_t *cref,
    PtWidget_t *wgt,
    PtGenTreeItem_t *item,
    PhEvent_t *event,
    int reason);
```

Arguments:

- `cref` The superclass whose method you want to call.
- `widget` A pointer to the widget for which to call the superclass’s method.
- `item` A pointer to a `PtGenTreeItem_t` structure (see the Photon Widget Reference) for the item to be redrawn.
- `event` A pointer to a `PhEvent_t` structure (see the Photon Library Reference) that describes the event.
- `reason` Either Pt_TREE_EXPANDING or Pt_TREE_COLLAPSING.

Description:

This function invokes the Tree Item State method of the class defined by `cref`.

Returns:

The value returned by the specified Tree Item State method.

Classification:

Photon

Safety

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

- `PtGenTree`, `PtGenTreeItem_t` in the Photon Widget Reference
- `PhEvent_t` in the Photon Library Reference
In this chapter...

Widget design considerations 139
Creating a shared object 140
Creating a template 140
Editing palette.def 140
Creating a widget description table 141
This chapter discusses adding custom widgets to PhAB, including widget design considerations as well as the steps to follow:

- Creating a shared object
- Creating a template
- Editing `palette.def`
- Creating a widget description table

Be sure to modify your application’s Makefile so that it’s linked with your new widget. Include your widget header in the global application header defined in the Application Start-up Info dialog.

**Widget design considerations**

This section describes the issues you need to consider when you add custom widgets to the PhAB widget palette:

- Single-value resources
- When to add custom widgets to palette files
- Displaying your custom widget

**Single-value resources**

PhAB has built-in editors for handling certain types of single-value resources. It also provides multiple-value editors for some resource types. If you design your resources to use the types supported by PhAB, developers will be able to use the widget in PhAB easily. If you define resources that have no associated editor in PhAB, you can create a resource editor for them as described in Creating custom resource editors, otherwise developers will be able to set and use these resources only through application code. The list of currently supported types can be found in the section on “Creating a widget description table.”

**When to add custom widgets to palette files**

When you define a widget class, you assign it a widget number. This number is combined with the resource number to create a series of unique resource numbers for the widget. When you add the widget to the PhAB widget palette, you define these resource values in the widget palette file.

PhAB uses these numbers when it stores the widget in the module files. If you change the resource number or widget number after it has been used in a PhAB application, you won’t be able to continue using the old widget (defined in the PhAB module) because the resource numbers will no longer match. For this reason, you should add widgets to the PhAB palette file only when almost all the widget’s functionality has been completed and the resource numbers will no longer change.
Displaying your custom widget

As described in “Editing palette.def,” later in this chapter, you can tell PhAB the name of the shared library for your widget. If PhAB can open this library, it can display your custom widget, and you’ll see the results of editing the widget’s resources.

If PhAB can’t open your widget library, you won’t be able to see the widget change visually when you make changes to the resources. The widget will display properly when you actually compile, link, and test the application.

Remember to include the widget header file in the application header so that the generated abmain.c file will know about your widget. Also, add your widget to the MYOBJ object list defined in the indOfiles file.

Creating a shared object

After you’ve created your widget, you need to build it into a shared object. To do this, compile your source code like this:

```c
cc -shared -o libmy_wgt.so my_wgt.c -v -Wl,-hlibmy_wgt.so
```

If you want to make a debug version of your widget library, add the compiler option `-gdwarf-2` to the command line before the `-shared` option. For more information about compile options, see qcc in the QNX Neutrino Utilities Reference.

You need to place this library in a directory that’s in the LD_LIBRARY_PATH path. You also need to put the header file for your custom widget in a place where you can find it later.

Creating a template

Use PhAB to create a template for your widget. For more information, see “Templates” in the Creating Widgets in PhAB chapter of the Photon Programmer’s Guide).

Editing palette.def

Next, edit the /usr/photon/appbuilder/palette.def file. You need to add some lines that look like this:

```c
l=PtWeb,<photon/PtWebClient.h>:ph
p=xpalette,Web Widgets
```

The first line is the library line, and starts with `l=`: It consists of:

- The library to load (e.g. PtWeb). PhAB adds `lib` to the start of the name, and `.so` to the end. In this example, the actual library is `libPtWeb.so`. Specify the name of the shared object that you created for your widget. If PhAB can open this library, it can display your widget “live;” and you’ll see the results of editing the resources immediately.
Creating a widget description table

The description table for a single widget is stored in a widget palette file. The widget description table must describe all the resources and callbacks your custom widget understands.

The palette file can contain any number of widget description tables but must contain at least one. The name of your palette file must end with a .pal extension (e.g. mywidget.pal). To get a good idea of what a palette file should contain, look at the /use/photon/appbuilder/ptpalette.pal file.

Here’s an example of a widget description table:

```
w=PwMenuWidget
h=4
PtWidget
PtBasic
PtContainer
PwMenuWidget

r=PW_LIST_DOWN,List Down Image,5001001,0,0,pixmap,NULL
r=PW_LIST_UP,List Up Image,5001004,0,0,pixmap,NULL
r=PW_SCROLL_DOWN,Scroll Down Image,5001005,0,0,pixmap,NULL
r=PW_SCROLL_UP,Scroll Up Image,5001006,0,0,pixmap,NULL
r=PW_SCROLL_LEFT,Scroll Left,5001007,0,0,pixmap,NULL
r=PW_SCROLL_RIGHT,Scroll Right, 5001008,0,0,pixmap,NULL

t=2
s=0xe906
```

The start of a new widget in the file is defined by a widget class name definition. This is followed by the widget hierarchy, resources, callbacks, type definition, and finally some initial default values to use when the widget is first created.

The widget description table defines the following settings:

- widget class name (w=)
- widget hierarchy (h=)
- list of resources (r=)
Creating a widget description table

- list of callbacks (c=, k=, e=)
- inherited resources and callbacks (i=, n=)
- base widget (b=)
- change class definition (q=)
- create-type definition (t=)
- cursor styles (s=)
- default values (d=)

**Widget class name (w=)**

This setting starts the definition of a new widget:

```plaintext
w=class_name
```

Example:

```plaintext
w=ShadowedBox
```

Don’t include spaces before or after the equals sign (=).

When PhAB encounters a class name setting, it allocates space in its internal list for the widget. Any settings found afterwards are applied to this widget until a new widget class name setting is encountered.

**Widget hierarchy (h=)**

This setting defines the widget hierarchy:

```plaintext
h=number_of_levels
highest_class_name
[next_highest_class_name]
::
custom_widget_class_name
```

Example:

```plaintext
h=3
PtWidget
PtBasic
ShadowedBox
```

The h= setting indicates the number of levels in the hierarchy. This is followed by the widget class hierarchy names in descending order of inheritance.

The widget hierarchy setting tells PhAB which class in the widget’s hierarchy to use when different classes of widgets are selected. For example, if you select a PtLabel and PtButton widget at the same time, the PhAB Control Panel must show you resources that apply to both widgets. PhAB does this by walking up the hierarchy list until it finds the identical class name for both widgets.
Creating a widget description table

List of resources (r=)

If you want to be able to edit a resource in PhAB, you’ll need either an r= entry (for new resources or inherited resources with a different default value) or an i= entry (for inherited resources) — see “Inherited resources and callbacks (i=, n=)” below.

The r= setting defines a resource for the widget:

```
r=manifest_name,desc_name,resource_no,reserved,select_ind, datatype[endian_string][abstract_name_of_editor],default_value[,additional_info][additional_info_1][additional_info_2][...][additional_info_n]
```

Example:

```
r=SBW_SHADOW_COLOR,Shadow Color,5000000,0,1,crgb,0x0
r=Pt_ARG_HORIZONTAL_ALIGNMENT,Horizontal Alignment,3000,0,1,choice,2,3
Pt_LEFT,0
Pt_RIGHT,1
Pt_CENTER,2
```

Don’t create resource entries for the Pt_ARG_POS and Pt_ARG_DIM resources. All widgets must support these, so PhAB handles them internally.

The arguments of the resource entry have the following meanings:

- **manifest_name**: The valid C manifest name to access this resource from application code.
- **desc_name**: A descriptive version of the manifest name, which is easier to understand and takes up less space (allowing the Control Panel to stay small).
- **resource_no**: The actual resource number (e.g. Pt_USER(0)=5000000, Pt_USER(1)=5001000, and so on — PtRESOURCE(Pt_USER(2),5)=5002005).
- **reserved**: Always use a value of 0.
- **select_ind**: Indicates whether this resource supports multiple selections. A value of 0 will hide the resource when two or more widgets are selected. A value of 1 enables the resource.
- **type**: Indicates the type of editor PhAB should use for this resource. For more detailed information about resource types, see Resource datatypes in the Creating Custom Resource Editors chapter. Valid types are:
  - **alloc**: A generic resource type, not covered by the other types.
Creating a widget description table

choice  An exclusive choice type.

code  A pointer to a function (see Pointer-to-function resources below).

crgb  A color value of type PgColor_t.

double  A double value.

flag  Flag resource.

font  Font specification (string).

list  Array of text strings.

multi  Multiline text string.

numeric  Numeric type.

pattern  A pattern type, for example Pt_ARG_FILL_COLOR.

pixmap  Supports the creation/modification of PhImage_t structures.

string  Single-line text string.

points  Used internally by PhAB for multisegment lines.

PhAB manages this resource type so you can use the points array provided by PtGraphic.

In addition, these types are supported for backwards compatibility, but are deprecated:

datas  Single-line text string (Alloc type). Deprecated; use string instead.

float  A float value. Deprecated; use double instead.

short  Numeric value. Deprecated; use numeric instead.

ushort  Unsigned value (a C type of unsigned short, unsigned long, unsigned int, or unsigned char). Deprecated; use numeric instead.

endian_string  Indicates the endian string for the resource. This is optional and is used only for alloc resource types.

Here’s an example of a data_type(endian_string)/abstract_name_of_editor string:

alloc(24css+il4c)/foo_editor

abstract_name_of_editor

Indicates the name of the editor for this resource, as defined in the res_editors.def file. PhAB parses a global res_editors.def file and a user specific res_editors.def file for the names of resource editors. See The res_editors.def file section of the Creating Custom Resource Editors chapter.

When abstract_name_of_editor, it is assumed to be the same as the data_type string ). Do not start your abstract_name_of_editor with a $. That’s reserved for PhAB use.
**default_value**

This is the default value in the widget code itself. It’s very important for this value to exactly match the real default value in the widget, because PhAB doesn’t generate resource entries in the module files if it seems the resource matches the default value. This keeps the size of the module file to a minimum. If the default value you specify doesn’t match, the widget may not behave as expected because a required resource setting will be missing from the module file.

If the resource datatype is `alloc`, you have to specify the total number of bytes for the default value, followed by a new line that contains the values. The values follow the endian string format. For instance, if your endian string is `2i3s` then you will have to specify 2 integer values followed by 3 short values. Each value is either a decimal value (for instance: `79`) or a hexadecimal value (for instance: `0x4f`). If the endian string is missing, the default endian string (`+c`) is used.

Each value is endian-less and they are separated by space or \n new lines.

Here’s an example:

```
r=Pt_ARG_MY_RESOURCE,My Resource,100220,0,0,alloc(11s)/my_resource,22
   0x0100 1 1 0 0 0 0 0 0 0 0 0
```

**additional_info**

This optional value can be used to specify additional information about the resource:

- A list of flag pairs for `flag`-type resources; see “Option and flag pairs” below.
- A list of option pairs for `short`-type resources; see “Option and flag pairs” below.
- A function prototype of `code`-type resources; see “Pointer-to-function resources” below.

The value can be specified as:

- The number of lines of additional information following the `r=` line.
  
  If there aren’t any lines of additional information, omit `additional_info`. Don’t specify a value of 0.

- The widget class for which the resource has been defined earlier, and from which you want to copy the additional information. This is useful, for example, if the resource’s description or default value is different for the two widgets, but the additional information is the same. If the resource is the same in both widgets, it’s easier to use an `i=` entry to make this widget inherit it from the other.
Option and flag pairs

For choice and flag resources, additional_info specifies the possible values. Each line is in the form:

description, value

where description is the text that PhAB is to display in the editor, and value is the corresponding value.

Pointer-to-function resources

For a resource that’s a pointer to a function (i.e. type is code), additional_info specifies the number of following lines that define the function’s prototype. The prototype should contain an @ in place of the function name, but no terminating semicolon.

The default_value specifies the code that will be used for the function body whenever PhAB generates a function for the resource. If the code will fit on the line and doesn’t contain any commas, you can specify it directly as default_value. Otherwise, put the code on the lines that follow the function prototype, and set default_value to be the number of lines of code.

Here are some examples:

```
r=ARG_SIMPLEFUNCTION,A function,5000001,5000001,1,return 0;,1
        int @( void )
```

```
r=Pt_ARG_RAW_DRAW_F,Draw Function,24000,24000,1,code,1,1
          void @(( PtWidget_t *widget, PhTile_t *damage )
                   PtSuperClassDraw( PtBasic, widget, damage );
```

```
r=Pt_ARG_LIST_BALLOON,Inflate Function,23031,23031,1,code,6,5
          PtWidget_t *@()
                   PtWidget_t *widget, PtWidget_t *parent,
                         PhArea_t *area, PtListColumn_t const *column, int coln,
                         const char *item, unsigned index, const char *font
                   )
          return
                   PtGenListCreateTextBalloon{
                         widget, parent,
                         PtGenListSetColumnBalloon( area, column ),
                         item, coln, font
                   };
```

List of callbacks (c=, k=, e=)

These settings define callbacks:

```
c | k | e=manifest_name,desc_name,resource_num,reserved
```

Example:

```
c=Pt_CB_ACTIVATE,Activate,2009,0
k=Pt_CB_HOTKEY,Hotkey,1010,0
e=Pt_CB_RAW,Raw Event,1011,0
```
Photon supports three different types of callbacks. These are standard widget callbacks (c=), hotkey callbacks (k=), and raw callbacks (e=). Make sure you use the type that matches the definition of the callback in the widget.

The arguments of the callback entries are identical and have the following meanings:

- **manifest_name**: The valid C manifest name to access this resource through application code.
- **desc_name**: A descriptive version of the manifest name, which is easier to understand and takes up less space (allowing the Control Panel to stay small).
- **resource_no**: The actual resource number (e.g. `Pt_USER(0)=5000000`, `Pt_USER(1)=5001000`, and so on — `PtRESOURCE(Pt_USER(2),5)=5002005`).
- **reserved**: Always use a value of 0.

Callbacks can also be inherited from other widgets. See “Inherited resources and callbacks (i=, n=)”, below.

**Inherited resources and callbacks (i=, n=)**

Resource descriptions can be shared by multiple widget classes. This saves memory on runtime and helps keep the palette files consistent (e.g. if a new flag is added to a resource of a parent class, you don’t have to add it to the description of each child class manually).

These settings can be used to inherit resource or callback definitions from a widget defined earlier:

```
i=widget[, from[, to]]
n=widget[, from[, to]]
```

The *i* line copies resources (defined by *r* or *i* lines). The *n* line copies callbacks (defined by *c*, *k*, *e*, or *i* lines).

The *from* and *to* values are numbers that specify the resource numbers to be copied. If both are given, they define a range. If only the *from* value is given, it specifies one resource. If neither *from* nor *to* is given, all resources/callbacks will be copied.

The *i* entry can be used only if the resource definitions for the child and parent classes are identical. If there are any differences (for example, the child class overrides the default value), you’ll need to use an *r* resource definition — but the child class can still inherit any additional_info that the parent class defines or inherits. For more information, see “List of resources (*r*=).”
Creating a widget description table

Base widget (b=)

This setting tells PhAB to use container emulation:

\[ b=\text{Container} \]

Since PhAB can’t display your widget directly, it must use a substitute widget in its place. If the widget isn’t a container and you don’t set this value, PhAB will use a \texttt{PtBasic} widget to emulate your widget. If the widget is a container, set the value as shown above and PhAB will use \texttt{PtContainer} to emulate your widget.

Change class definition (q=)

This definition table setting limits the classes the current container widget can be changed into, if it contains children:

\[ q=\text{widgetclass[, widgetclass]} \]

Example:

\[ q=\text{PtContainer,PtGenList} \]

The change class definition tells PhAB which classes this container widget class may be changed into, if it contains children widgets. This operation is performed when a user selects the \texttt{Change Class} command in PhAB.

The general rule is, if a widget class has a setting of \texttt{q=a,b,c}, it can only be class changed into a class that has a setting of \texttt{q=a...}, \texttt{q=b...}, or \texttt{q=c...}. That is, the first item of the target class’s \texttt{q=} setting must match a widget from the \texttt{q=} setting of the original class.

For example, all list and tree widgets have the setting \texttt{q=PtGenList,PtContainer}, and \texttt{PtPane} has \texttt{q=PtContainer}. This means you can change from a list with a child into a \texttt{PtPane}, but not from a \texttt{PtPane} with a child into a list.

If the change class definition \texttt{q=} is missing from a widget class’s definition table, then that widget class cannot be used as a target when the original widget has children.

If a container widget has no children, it can be changed into any other type of widget.

Create-type definition (t=)

This setting defines the type of creation mode to use:

\[ t=\text{number} \]

Example:

\[ t=2 \]

The type definition tells PhAB how this widget should be created. For example, if \texttt{t=1}, PhAB automatically creates the widget as soon as the user clicks in the module. This is because type 1 indicates the widget has a preset size. A value of \texttt{t=2} means the user can drag out the size when creating the widget.

Valid values for \texttt{number} range from 1 through 6 and have the following meanings:
When number is: | Create type is:
---|---
1 | Preset, resizable
2 | Resizable on create
3 | Line (2 points)
4 | Preset, Nonresizable
5 | Polygon (multipoint)
6 | Bezier (multipoint with handles)

For more information, see “Creating a widget” in the Creating Widgets in PhAB chapter of the Photon *Programmer’s Guide*.

**Cursor styles (s=)**

This setting determines the cursor style when creating the widget:

\[ s=start\_cursor\_style,[drag\_cursor\_style] \]

Example:

\[ s=0xe914,0xe914 \]

This setting defines the look of the cursor when the widget is being created in PhAB. The value you choose for a cursor style setting (s=) will depend on the value you specify for the create type setting (t=). For consistency with other widgets in PhAB, we recommend that you assign cursor style values according to the table below:

<table>
<thead>
<tr>
<th>If create type is:</th>
<th>Cursor style should be:</th>
</tr>
</thead>
<tbody>
<tr>
<td>t=1</td>
<td>s=0xe906</td>
</tr>
<tr>
<td>t=2</td>
<td>s=0xe914,0xe914</td>
</tr>
<tr>
<td>t=3</td>
<td>s=0xe906</td>
</tr>
<tr>
<td>t=4</td>
<td>s=0xe906</td>
</tr>
<tr>
<td>t=5</td>
<td>s=0xe906</td>
</tr>
<tr>
<td>t=6</td>
<td>s=0xe906</td>
</tr>
</tbody>
</table>

**Default values (d=)**

This setting lets you give the widget some default values to override the defaults built into the widget:

\[ d=class\_name,no\_of\_resource\_definitions \]

\[ definition\_1 \]

\[ definition\_2 \]
Creating a widget description table

Each definition consists of three values:

- resource_number
- resource_type
- resource_value

For example:

// Resizable example.
d=ShadowedBox,1
1005
dim
1,1

// Preset example.
d=PtPrintSel,1
1005
dim
418,285

Each resource definition is made up of three lines, and only simple resource types can be used. In the example above, the Pt_ARG_DIM (1005) resource is set to 1 pixel high and 1 pixel wide. This is a good starting point for a widget with a create type number of 2.

**CAUTION:** You must always give a widget a default value and starting dimension. If you don’t, you’ll get unexpected results.

If t=1 or t=4, set the dim resource to the preset size. If the value is anything else, set dim to 1,1. The dim resource is defined as “width,height” in pixels.
This chapter contains descriptions of the following functions, which are included in the widget building library API:

- **PhWindowChange()**  Modify the attributes of a window’s region
- **PhWindowClose()**  Close a window
- **PhWindowOpen()**  Create a window region
- **PtAddWidgetData()**  Add data to the widget data chain
- **PtAnchorDeregister()**  Deregister a widget from its parent for anchoring
- **PtAnchorRegister()**  Register a widget with its parent for anchoring
- **PtAnchorWidget()**  Anchor the provided widget
- **PtApplyAnchors()**  Anchor a widget and its children
- **PtAttemptResize()**  Adjust the size of a widget
- **PtCalcAnchorOffsets()**  Update the anchoring values (rules) for the given widget
- **PtCalcRegion()**  Determine whether or not a widget needs a region
- **PtChildBoundingBox()**  Calculate a widget’s canvas and its children’s bounding boxes
- **PtClipAdd()**  Add a clipping rectangle to the stack
- **PtClipRemove()**  Take a clipping rectangle off the stack
- **PtCompoundRedirect()**  Redirect widgets to a parent
- **PtCoreChangeRegion()**  Determine if a region is required
- **PtCreateWidgetClass()**  Create a widget class
- **PtDamageExposed()**  Damage the specified widgets
- **PtDestroyCallbackList()**  Free the specified callbacks
- **PtDestroyHotkeyCallbacks()**  Free the specified hotkey callbacks
\textit{PtDestroyRawCallbacks()}
Free the specified raw callbacks

\textit{PtFindNextWidgetData()}
Find the next appropriate data block

\textit{PtFindResource()}
Find the record associated with a resource

\textit{PtFindResourceRecord()}
Deprecated. Use \textit{PtFindResource()}

\textit{PtFindWidgetData()}
Find the first data block of a given type and subtype

\textit{PtGetCallbackList()}
Get a callback list

\textit{PtGetStruct()}
Retrieve the specified resource

\textit{PtInvokeCallbackList()}
Invoke a callback list

\textit{PtInvokeCallbackType()}
Invoke a callback list of a specific type

\textit{PtInvokeResizeCallbacks()}
Invoke the resize callbacks of the specified container

\textit{PtMoveResizeWidget()}
Synchronize a widget’s extent

\textit{PtRemoveWidgetData()}
Remove data from the widget data chain

\textit{PtResizeCanvas()}
Set the size of a widget’s canvas

\textit{PtResizePolicy()}
Determine whether a widget has a resize policy

\textit{PtSetExtentFromArea()}
Calculate the extent of a widget

\textit{PtSetStruct()}
Set the specified resource

\textit{PtSetValue()}
Set the value of a resource using \texttt{mod_f}

\textit{PtSuperClassCalcOpaque()}
Call the Calc Opaque Rect method of the specified superclass

\textit{PtSuperClassChildCreated()}
Invoke a Child Created method

\textit{PtSuperClassChildDestroyed()}
Invoke a Child Destroyed method
PtSuperClassChildGettingFocus()
    Invoke a Child Getting Focus method

PtSuperClassChildGettingResources()
    Invoke a Child Getting Resources method

PtSuperClassChildLosingFocus()
    Invoke a Child Losing Focus method

PtSuperClassChildMovedResized()
    Invoke a Child Moved/Resized method

PtSuperClassChildRealized()
    Invoke a Child Realized method

PtSuperClassChildSettingResources()
    Invoke a Child Setting Resources method

PtSuperClassChildUnrealized()
    Invoke a Child Unrealized method

PtSuperClassConnect(), PtSuperClassConnectFrom()
    Invoke the Connection method of the specified widget class

PtSuperClassDraw()
    Invoke the Draw method of the specified superclass

PtSuperClassExtent()
    Invoke the Extent method of the specified superclass

PtSuperClassGetResources()
    Get the specified resource

PtSuperClassGotFocus()
    Invoke the Got Focus method of the specified superclass

PtSuperClassInit(), PtSuperClassInitFrom()
    Invoke the Initialize method of the specified widget class

PtSuperClassLostFocus()
    Invoke the Lost Focus method of the specified superclass

PtSuperClassRawEvent(), PtSuperClassRawEventFrom()
    Invoke the raw callback list of the specified widget class

PtSuperClassRealized()
    Invoke the Realization method of the specified widget class
PtSuperClassSetResources()

Set resources

PtUpdateVisibility()  Tell the widget library about a change in visibility

PtWidgetAbove()  Get the widget that’s above a given widget in a family hierarchy
PhWindowChange()
Modify the attributes of a window’s region

Synopsis:

```
int PhWindowChange(
    unsigned fields,
    unsigned flags,
    const PhRegion_t *region,
    const PhRect_t *rect,
    const PhWindowInfo_t *win_info);
```

Arguments:

- `fields`: A bitmap that indicates which members of the `PhRegion_t` to use to update the window’s region. For more information, see `PhRegionOpen()` in the Photon Library Reference.

- `flags`: Flags that control whether or not an expose event is emitted to the region, when necessary. You can OR the following into `flags`:
  - `Ph_EXPOSE_REGION` — if part of the region becomes exposed, send a `Ph_EV_EXPOSE` event to the region.
  - `Ph_EXPOSE_FAMILY` — if part of the region becomes exposed, send a `Ph_EV_EXPOSE` event to the region’s descendants.

- `region`: A pointer to a `PhRegion_t` structure that stores the data to be used to update the window’s region.

- `rect`: A pointer to a `PhRect_t` structure (see the Photon Library Reference) that defines the rectangle associated with the region.

- `win_info`: A pointer to the window information that replaces the current `win_info` on the window region.

Library:

```
ph
```

Description:

This function lets you modify the attributes and window information of a window’s region. The Window Manager is notified of this change and responds accordingly.

Don’t use this function in an application that uses widgets.

This function changes the definition of the window region specified by `region->rid`.

Returns:

A nonnegative value
  - Successful completion.

-1 An error occurred.
Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PhWindowOpen()

PhRect_t, PhRegion_t, PhRegionClose(), PhRegionOpen() in the Photon Library Reference

See PhWm.h for a description of PhWindowInfo_t.
PhWindowClose()

Close a window

Synopsis:

```c
int PhWindowClose( PhRid_t window_rid );
```

Arguments:

- `window_rid` The region ID of the window to close.

Library:

```
ph
```

Description:

This function closes a window with the given region ID, that was previously opened by a call to `PhWindowOpen()`.

Don’t use this function in an application that uses widgets.

Returns:

- A nonnegative value
  - Successful completion.
- `-1` An error occurred. Either a nonwindow RID was given, or the function can’t communicate with Photon or the window manager.

Classification:

```
Photon
```

Safety

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

`PhWindowOpen()`
PhWindowOpen()  

Create a window region

Synopsis:

```c
PhRid_t PhWindowOpen(
    unsigned fields,
    const PhRegion_t *region,
    const PhRect_t *rect,
    const PhWindowInfo_t *win_info );
```

Arguments:

- `fields`: A bitmap that indicates which members of the `PhRegion_t` structure specified by `region` to use. For more information, see `PhRegionOpen()` in the Photon Library Reference. If you don’t specify a field, `PhWindowOpen()` sets the corresponding member of the new region to its default value.

- `region`: A pointer to a `PhRegion_t` structure that’s used as a template when opening the new region. You must set the `parent` member of `region`; Photon fills in the other family members.

- `rect`: A pointer to a `PhRect_t` structure (see the Photon Library Reference) that defines the rectangle associated with the region.

- `win_info`: A pointer to a description of the window’s attributes, as specified in the `PhWindowInfo_t` structure (see `PhWm.h`).

Library:

- `ph`

Description:

This function creates a window region that will be managed by the Window Manager in the Photon event space. The Window Manager will provide a region to act as a frame in addition to the region created by the calling application.

Don’t use this function in an application that uses widgets.

Returns:

The region ID of the new window’s interior (not the RID of the frame).

If an error occurred, this function returns:

- `-1`: The function couldn’t communicate with Photon.

- `0`: The region couldn’t be created (e.g. the system is out of memory).
PhWindowOpen()

Classification:

Photon

Safety

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PhWindowChange(), PhWindowClose()

PhRect_t, PhRegion_t, PhRegionChange(), PhRegionOpen() in the Photon Library Reference

See PhWm.h for a description of PhWindowInfo_t.
Add data to the widget data chain

Synopsis:

```c
int PtAddWidgetData( PtWidget_t *widget,
                     PtWidgetClassRef_t *type,
                     long subtype,
                     void *data );
```

Arguments:

- `widget` A pointer to the widget whose chain the data should be added to.
- `type` The class of the widget adding the data.
- `subtype` A subtype that’s used to discriminate between multiple blocks of data added by the same widget class. The `subtype` shouldn’t be -1, as this value has special meaning when searching for a specific block within the widget data chain.
- `data` A pointer to the data to be added to the widget data chain.

Description:

This function adds a piece of data to the widget data chain. The data provided must be in a block of memory allocated by `malloc()` (see the QNX Neutrino Library Reference).

You can retrieve this data by calling `PtFindWidgetData()` or `PtFindNextWidgetData()`.

Returns:

0 on success; -1 if an error occurred (e.g. out of memory).

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

See also:

`PtFindWidgetData()`, `PtFindNextWidgetData()`, `PtRemoveWidgetData()`
PtAnchorDeregister()
Deregister a widget from its parent for anchoring

Synopsis:

void PtAnchorDeregister( PtWidget_t *widget );

Arguments:

widget      A pointer to the widget that’s to be deregistered.

Description:

This function deregisters \texttt{widget} from its parent. This prevents \texttt{widget} from being included in anchoring operations.

The deregister operation is done automatically when the anchor flags are cleared.
You’ll need to use this function only if you take over the \texttt{Pt_ARG_ANCHOR_FLAGS} and \texttt{Pt_ARG_ANCHOR_OFFSETS} resources.

Classification:

Photon

\begin{tabular}{|l|c|}
\hline
\textbf{Safety} & \\
\hline
Interrupt handler & No \\
Signal handler & No \\
Thread & No \\
\hline
\end{tabular}

See also:

\textit{PtAnchorRegister()}
Synopsis:

```c
void PtAnchorRegister( PtWidget_t *widget );
```

Arguments:

`widget` A pointer to the widget to register.

Description:

This function registers `widget` with its parent for anchoring. Registration is done automatically when the anchor flags are set. You’ll need to use this function only if you take over the `Pt_ARG_ANCHOR_FLAGS` and `Pt_ARG_ANCHOR_OFFSETS` resources.

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

`PtAnchorDeregister()`
PtAnchorWidget()
Anchor the provided widget

Synopsis:

```c
void PtAnchorWidget( PtWidget_t *widget );
```

Arguments:

`widget`  A pointer to the widget to anchor.

Description:

This function anchors the provided `widget`. If the widget’s anchor offsets have yet to be calculated, they’re calculated along with the anchor offsets for all children. No other action takes place.

If the widget’s anchor offsets have been calculated, the widget and all its children’s dimensions are adjusted to honor their anchor flags.

---

`PtAnchorWidget()` is used by `PtApply Anchors()`.

---

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

`PtApplyAnchors()`, `PtCalcAnchorOffsets()`
Anchor a widget and its children

**Synopsis:**

```c
int PtApplyAnchors( PtWidget_t *widget );
```

**Arguments:**

- `widget` A pointer to the widget to be anchored.

**Description:**

This function performs any necessary anchoring on the provided `widget` and all its registered children.

**Returns:**

- 1 Success.
- 0 An error occurred.

**Classification:**

- Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

**See also:**

- `PtAnchorDeregister()`, `PtAnchorRegister()`, `PtAnchorWidget()`
Synopsis:

```c
int PtAttemptResize( PtWidget_t *widget,
    PhRect_t const *canvas,
    PhRect_t const *render );
```

Arguments:

- `widget`: A pointer to the widget whose size you want to adjust.
- `canvas`: A pointer to a `PhRect_t` structure (see the Photon Library Reference) that describes the widget’s canvas.
- `render`: A pointer to a `PhRect_t` structure that defines the part of the widget you want to render.

Description:

This function adjusts the size of the widget based on the differences between `canvas` and `render` and on the widget’s resize flags.

`PtResizeCanvas()` is similar to this function but easier to use. You should call it instead of `PtAttemptResize()`.

The widget’s actual size is modified (`widget->area.size`). If the resize policy of the widget prevents `PtAttemptResize()` from adjusting the widget’s size (i.e. the canvas won’t fit within the provided render rectangle), `PtAttemptResize()` sets the Pt_UCLIP bit of the widget’s resize flags. If this bit is set, the widget’s Draw method should apply clipping via `PtClipAdd()` prior to rendering its data.

Returns:

- `1`: Successful completion.
- `0`: No resize occurred.

Classification:

- Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th>Interrupt handler</th>
<th>Signal handler</th>
<th>Thread</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>
See also:

PtChildBoundingBox(), PtClipAdd(), PtClipRemove(), PtResizeCanvas()
PgExtentText(), PhRect_t, PtCalcCanvas() in the Photon Library Reference
Synopsis:

```c
int PtCalcAnchorOffsets( PtWidget_t *widget );
```

Arguments:

`widget` A pointer to the widget whose anchoring values you want to update.

Description:

This function updates the anchoring rules for the given `widget` based on its current anchor flags and its parent’s canvas rectangle.

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

`PtAnchorWidget()`, `PtApplyAnchors()`
Determine whether or not a widget needs a region

**Synopsis:**

```c
int PtCalcRegion( unsigned int *fields,
                    PtWidget_t *widget,
                    PhRegion_t *region,
                    PhRect_t *rect );
```

**Arguments:**

- `fields` A bitmap that specifies which fields of the `PhRegion_t` structure pointed to by `region` the function should fill in. For more information, see `PhRegionOpen()` in the Photon Library Reference.
- `widget` A pointer to the widget you want to test.
- `region` A pointer to a `PhRegion_t` structure in which the function stores information about the region.
- `rect` A pointer to a `PhRect_t` structure that defines the rectangle for the region.

**Description:**

This function determines whether or not the given `widget` needs a region and stores information about the region in the `PhRegion_t` structure pointed to by `region`.

**Returns:**

- 0 The widget doesn’t need a region.
- 1 The widget needs a region.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

`PhRect_t`, `PhRegion_t`, `PhRegionOpen()` in the Photon Library Reference
**PtChildBoundingBox()**

*Calculate a widget’s canvas and its children’s bounding boxes*

**Synopsis:**

```c
PhRect_t *PtChildBoundingBox( PtWidget_t *widget,
                            PhRect_t *canvas,
                            PhRect_t *render );
```

**Arguments:**

- `widget`  
  A pointer to the widget for which you want to calculate the canvas.

- `canvas`  
  A pointer to a `PhRect_t` structure (see the Photon Library Reference). If  
  this pointer isn’t NULL, the rectangle it points to is set to the canvas  
  rectangle of the `PtBasic`-class level for the specified `widget`.

- `render`  
  A pointer to a `PhRect_t` structure in which the function stores the  
  bounding box. You must provide a non-NULL pointer for this argument.

**Description:**

This function calculates the canvas of `widget` and finds the bounding box of `widget`’s  
children relative to the origin of the parent’s position.

**Returns:**

The same pointer as the `render` argument.

**Classification:**

Photon

---

**Safety**

<table>
<thead>
<tr>
<th>Safety</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

- `PtSuperClassExtent()`
- `PhRect_t`, `PtCalcCanvas()` in the Photon Library Reference
Add a clipping rectangle to the stack

Synopsis:

```c
int PtClipAdd( PtWidget_t *widget,
               PhRect_t *rect );
```

Arguments:

- `widget`: A pointer to the widget being clipped.
- `rect`: A pointer to a `PhRect_t` structure (see the Photon Library Reference) that specifies the clipping to be added.

Description:

This function adds a clipping rectangle to the clipping stack. The rectangle added to the clipping stack is the intersection of the last rectangle on the stack and the provided rectangle `rect`.

Prior to entering a widget’s Draw method, the canvas rectangle derived from the `PtBasic`-class level is pushed onto the clipping stack. This prevents any children from drawing beyond the canvas of the parent container.

A widget can, however, draw beyond its own canvas or extent unless additional clipping is performed. `PtAttemptResize()` and `PtResizeCanvas()` set the Pt_UCLIP bit of a widget’s resize flags if the widget requires additional clipping (to prevent it from drawing beyond its own canvas).

Returns:

The current level of stack clipping.

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

`PtAttemptResize()`, `PtClipRemove()`, `PtResizeCanvas()`

`PhRect_t`, `PtCalcCanvas()` in the Photon Library Reference
**PtClipRemove()**

*Take a clipping rectangle off the stack*

**Synopsis:**

```c
int PtClipRemove();
```

**Description:**

This function pops the last clipping rectangle off the clipping stack.

**Returns:**

The current level of stack clipping.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

*PtAttemptResize(), PtClipAdd(), PtResizeCanvas()*

*PtCalcCanvas() in the Photon Library Reference*
Redirect widgets to a parent

Synopsis:

```c
PtWidget_t *PtCompoundRedirect(
    PtWidget_t *container,
    PtWidgetClassRef_t *child_cref);
```

Arguments:

- `container`: A pointer to the container widget from which you want to redirect the children.
- `child_cref`: The class of the child widget (e.g. `PtButton`).

Description:

This function returns the parent of a compound widget, making the compound widget seem like a simple widget (i.e. users can’t place widgets inside `container`).

This is especially useful for rejecting widgets or redirecting widgets to subordinate containers (e.g. `PtScrollArea` does this).

This redirector function can be added to a custom compound widget by using the `Pt_SET_CHILD_REDIRECT_F` manifest defined by `PtContainer`.

Returns:

A pointer to the parent of the given compound widget.

Classification:

Photon

Safety

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

“Container widget anatomy” in the Anatomy of a Widget chapter.
PtCoreChangeRegion()
Determine if a region is required

Synopsis:

```c
void PtCoreChangeRegion( unsigned int fields,
                         PtWidget_t *widget );
```

Arguments:

- `fields`: A bitmap that indicates which attributes of the widget to check. For more information, see `PhRegionOpen()` in the Photon Library Reference.
- `widget`: A pointer to the widget to be tested.

Description:

This function examines the current state of `widget` with respect to `fields` and determines if `widget` requires a region. If a region is required, `PtCoreChangeRegion()` determines what attributes (e.g., sensitivity) are required.

`PtCoreChangeRegion()` may call one of the following functions with appropriate attributes:

<table>
<thead>
<tr>
<th>If the widget:</th>
<th>This function is called:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Is a subclass of <code>PtWindow</code></td>
<td><code>PhWindowChange()</code></td>
</tr>
<tr>
<td>Already has a region</td>
<td><code>PhRegionChange()</code></td>
</tr>
<tr>
<td>Doesn’t have a region</td>
<td><code>PhRegionOpen()</code></td>
</tr>
</tbody>
</table>

Classification:

- Photon

Safety

<table>
<thead>
<tr>
<th></th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td></td>
</tr>
<tr>
<td>Signal handler</td>
<td></td>
</tr>
<tr>
<td>Thread</td>
<td></td>
</tr>
</tbody>
</table>

See also:

- `PhWindowChange()`
- `PhRegionChange()`, `PhRegionOpen()` in the Photon Library Reference
### Synopsis:

```c
PtWidgetClass_t *PtCreateWidgetClass( 
    PtWidgetClassRef_t *superclass_ref, 
    unsigned int size, 
    unsigned int num_args, 
    PtArg_t const *args );
```

### Arguments:

- **superclass_ref**
  - The superclass for the new widget class.

- **size**
  - The size of the new widget class. If this is 0, the size of the superclass is used; if nonzero, it must be at least the size of the superclass.

- **num_args**
  - The number of entries in the `arg` array.

- **args**
  - An array of class attributes and member functions to be applied when creating the new class.

### Description:

This function creates a new widget class based on `superclass_ref`. If the specified superclass hasn’t yet been created, it’s created now.

If the `size` parameter is zero, the new class is the same size as the specified superclass. Otherwise `size` bytes are allocated for the new class. The specified superclass is then copied into the newly allocated class (inheritance). The `version`, `resources`, `num_resources`, `callbacks`, `dflts_f`, `connect_f`, `init_f`, `unrealize_f`, and `destroy_f` members are cleared because they shouldn’t be inherited.

### Returns:

A pointer to the newly created class:

```c
base->superclass = superclass ? superclass->wclass:NULL;
```

### Examples:

This example is from the `ShadowedBox` sample widget.

```c
// // ShadowedBox class-creation function
//
PtWidgetClass_t *PtCreateBasicClass( void )
{
    static const PtResourceRec_t resources[] =
    {
        { SBW_SHADOW_COLOR, Pt_CHANGE_REDRAW, 0,
            Pt_ARG_IS_NUMBER( ShadowedBoxWidget, shadow_color ) },
        { SBW_SHADOW_OFFSET, Pt_CHANGE_REDRAW, 0,
            Pt_ARG_IS_NUMBER( ShadowedBoxWidget, shadow_offset ) }
    }
```
};

static const PtArg_t args[] =
{
    { Pt_SET_VERSION, 110 },
    { Pt_SET_STATE_LEN, sizeof( ShadowedBoxWidget ) },
    { Pt_SET_DFLTS_F, (long)shadowbox_dflts },
    { Pt_SET_DRAW_F, (long)shadowedbox_draw },
    { Pt_SET_FLAGS, 0, Pt_RECTANGULAR },
    { Pt_SET_RESOURCES, (long) resources },
    { Pt_SET_NUM_RESOURCES,
        sizeof( resources )/sizeof( resources[0] ) }
};

return( ShadowedBox->wclass = PtCreateWidgetClass(
    PtBasic, 0, sizeof( args )/sizeof( args[0] ), args ) );

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

See also:

PtArg_t in the Photon Library Reference
PtDamageExposed()

Damage the specified widgets

Synopsis:

```c
void PtDamageExposed( PtWidget_t *widget,
                      PhTile_t *tile );
```

Arguments:

- `widget` A pointer to the widget to damage.
- `tile` A list of `PhTile_t` structures (see the Photon Library Reference) that define the rectangles to damage. These rectangles should be relative to the widget’s origin.

Description:

This function damages the list of rectangles given in `tile` on the parent of `widget`. The damage may extend beyond the extent of the widget.

This function frees the tile list automatically.

Examples:

To redraw a transparent widget due to a data change:

```c
tile = PhGetTile();
PtWidgetExtent( widget, &tile->rect );
tile->next = NULL;
PtDamageExposed( widget, tile );
```

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

`PhGetTile()`, `PhTile_t` in the Photon Library Reference
Synopsis:

```c
int PtDestroyCallbackList(  
    PtCallbackList_t **cbp );
```

Arguments:

`cbp`  A pointer to the callback list to be freed.

Description:

This function frees the entire callback list pointed to by `cbp`. The `cbp` argument is set to NULL when `PtDestroyCallbackList()` is successful.

The widget library automatically frees all allocated resources including callback lists when a widget is destroyed.

Returns:

0  Successful completion.

-1  An error occurred.

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

See also:

`PtDestroyHotkeyCallbacks()`, `PtDestroyRawCallbacks()`
Free the specified hotkey callbacks

Synopsis:

```c
void PtDestroyHotkeyCallbacks( PtWidget_t *widget );
```

Arguments:

- `widget`: A pointer to the widget whose hotkey callbacks you want to free.

Description:

This function frees all hotkey callbacks connected to `widget`.

The widget library automatically frees all allocated resources including lists when a widget is destroyed.

Classification:

- Photon
- Safety
  - Interrupt handler: No
  - Signal handler: No
  - Thread: No

See also:

- `PtDestroyCallbackList()`
- `PtDestroyRawCallbacks()`
PtDestroyRawCallbacks()
Free the specified raw callbacks

Synopsis:

```c
void PtDestroyRawCallbacks( PtWidget_t *widget );
```

Arguments:

- `widget` A pointer to the widget whose raw callbacks you want to free.

Description:

This function frees all raw callbacks connected to `widget`.

> The widget library automatically frees all allocated resources including callback lists when a widget is destroyed.

Classification:

- Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

- `PtDestroyCallbackList()`, `PtDestroyHotkeyCallbacks()`
**PtFindNextWidgetData()**

*Find the next appropriate data block*

### Synopsis:

```c
void * PtFindNextWidgetData(
    PtWidget_t *widget,
    PtDataHdr_t *data_node,
    PtWidgetClassRef_t *type,
    long subtype,
    PtDataHdr_t **_node )
```

### Arguments:

- `widget` A pointer to the widget whose data you want to search.
- `data_node` A pointer to the data block from which to start the search. If `data_node` is NULL, the function searches for the first instance of data that matches the type and subtype provided.
- `type` The type of data to seek. If `type` is NULL, any type matches.
- `subtype` The subtype of data to seek. If `subtype` is -1, any subtype matches.
- `_node` If you provide a non-NULL pointer for this argument, it’s set to point to the widget `data_node` that contained the returned data, so you can continue the search from that node.

### Description:

This function (starting from `data_node`) finds the next widget data block that matches the `type` and `subtype` provided.

### Returns:

A void pointer to the widget data, or NULL if it wasn’t found.

### Classification:

- **Photon**

  **Safety**
  
<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

### See also:

- `PtAddWidgetData()`, `PtFindWidgetData()`, `PtRemoveWidgetData()`
PtFindResource()
Find the record associated with a resource

Synopsis:

```c
PtResourceRec_t const * PtFindResource(
    long type,
    PtWidgetClass_t const * const a_class);
```

Arguments:

- `type` The resource you want to find (e.g. Pt_ARG_COLOR).
- `a_class` The widget class to search.

Description:

This function finds the resource record for the resource given by `type` in the `a_class` class or any of its superclasses. You can use this function to detect whether a widget implements a particular resource or not, or in your own Set Resource/Get Resource methods.

Returns:

A pointer to a resource record, or NULL if no resource is found.

Examples:

```c
my_label_setcolor( PtWidget_t *widget, PgColor_t color )
{
    const PtResourceRec_t *res_rec;
    PtArg_t argt;
    PtSetArg( &argt, Pt_ARG_COLOR, color, 0 );
    res_rec = PtFindResource( Pt_ARG_COLOR,
                             PtLabel->wclass );
    PtSetValue( widget, res_rec, &argt );
}
```

Classification:

- Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
See also:

PtSetValue()
**PtFindWidgetData()**

Find the first widget data block of a given type and subtype

### Synopsis:

```c
void * PtFindWidgetData(
    PtWidget_t *widget,
    PtWidgetClassRef_t *type,
    long subtype,
    PtDataHdr_t **_node);
```

### Arguments:

- **widget**
  A pointer to the widget that you want to search.

- **type**
  The type of data to seek. If `type` is NULL, any type matches.

- **subtype**
  The subtype of data to seek. If `subtype` is -1, any subtype matches.

- **_node**
  If you provide this argument, it’s set to point to the widget data node that contained the returned data, so you can continue the search from that node.

### Description:

This function finds the first widget data block that matches the `type` and `subtype` provided.

### Returns:

A void pointer to the widget data, or NULL if it wasn’t found.

### Classification:

**Photon**

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

### See also:

`PtAddWidgetData()`, `PtFindNextWidgetData()`, `PtRemoveWidgetData()`
Get a callback list

Synopsis:

```c
void *PtGetCallbackList( PtWidget_t *widget,
                        long cb_type );
```

Arguments:

- `widget`: The widget pointer to get the callback list from.
- `cb_type`: The callback list type to return.

Description:

This function returns the requested callback list of `cb_type`.

Returns:

A pointer to a `PtCallbackList_t` if the callback list type exists, NULL otherwise.

Examples:

```c
PtCallbackList_t *cb;
if( (cb = PtGetCallbackList( widget, Pt_CB_ARM) ) {
    // do the work to create and fill in the callback data ...
    PtInvokeCallbackList( cb, widget, &cbinfo );
}
```

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

`PtInvokeCallbackList()`, `PtInvokeCallbackType()`, `PtCallbackInfo_t` in the Photon Widget Reference
PtGetStruct()

Retrieve the specified resource

Synopsis:

```c
int PtGetStruct( char *base,
                 PtResourceRec_t const *mod,
                 PtArg_t *arg );
```

Arguments:

- `base` The base address in the widget from which to fetch the resource.
- `mod` The resource record for the resource.
- `arg` A pointer to a PtArg_t structure (see the Photon Library Reference) that describes the resource you want to retrieve. For more information, see below.

Description:

This function retrieves the resource specified by `arg` from the widget at address `base` as described by `mod`.

If the `value` and `len` members of `arg` are non-NULL, they’re assumed to contain the addresses of pointers of the type appropriate for the resource. The pointers are set to point to the resource’s value in the widget’s internal memory.

Don’t use these pointers to change the resource’s value.

If either of the `value` and `len` members of `arg` is NULL, the retrieved resources are stored in these members.

Returns:

- 0 Successful completion.
- -1 An error occurred.

Classification:

Photon

Safety

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
See also:

`PtSetStruct()`, `PtSuperClassGetResources()`, `PtSuperClassSetResources()`

`PtArg_t` in the Photon Library Reference
PtInvokeCallbackList()

Invoke a callback list

Synopsis:

```c
int PtInvokeCallbackList( PtCallbackList_t *cb_list,
    PtWidget_t *widget,
    PtCallbackInfo_t *cbinfo );
```

Arguments:

- `cb_list` The list of callbacks to invoke.
- `widget` The widget pointer to pass to the callbacks as the first argument.
- `cbinfo` A pointer to a `PtCallbackInfo_t` structure (see the Photon Widget Reference) that’s passed to each callback in the list as the third argument.

Description:

This function invokes the provided callback list `cb_list`.

The `cb_data` member of each item in the callback list is passed as the second argument to the associated callback.

Returns:

A return status from the callback list:

- `Pt_CONTINUE`
- `Pt_HALT`
- `Pt_END`

If the returned status is `Pt_END`, have your function consume the event (i.e. return `Pt_END`).

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

See also:

- `PtGetCallbackList()` and `PtInvokeCallbackType()`
- `PtCallbackInfo_t` in the Photon Widget Reference
Invoke a callback list of a specific type

Synopsis:

```c
int PtInvokeCallbackType( PtWidget_t *widget,
                        long type,
                        PtCallbackInfo_t *cbinfo ) ;
```

Arguments:

- `widget` The widget pointer to pass to the callbacks as the first argument.
- `type` The type of callback list to invoke, for example `Pt_CB_ARM`.
- `cbinfo` A pointer to a `PtCallbackInfo_t` structure (see the Photon Widget Reference) that’s passed to each callback in the list as the third argument.

Description:

This function invokes a callback list of `type` for a `widget`. It’s a convenience function that you can use instead of `PtGetCallbackList()` and `PtInvokeCallbackList()`.

Returns:

A return status from the callback list:

- `Pt_CONTINUE`
- `Pt_HALT`
- `Pt_END`

If the returned status is `Pt_END`, have your function consume the event (i.e. return `Pt_END`).

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

- `PtGetCallbackList()` and `PtInvokeCallbackList()`
- `PtCallbackInfo_t` in the Photon Widget Reference
Synopsis:

```c
void PtInvokeResizeCallbacks( PtWidget_t *container );
```

Arguments:

- `container` A pointer to the container widget whose resize callbacks you want to invoke.

Description:

This function sets up a `cbinfo` structure and invokes the resize callbacks attached to the specified `container`.

Classification:

- **Safety**
  - Interrupt handler: No
  - Signal handler: No
  - Thread: No
PtMoveResizeWidget()
Synchronize a widget's extent

Synopsis:

```c
int PtMoveResizeWidget( PtWidget_t *widget,
unsigned blit );
```

Arguments:

- `widget` A pointer to the widget whose extent you want to synchronize.
- `blit` A flag that indicates whether or not to use blitting; one of:
  - 0 — don’t blit.
  - Pt_BLIT_FORCE — always blit (this may cause unexpected effects).
  - Pt_BLIT — blit when possible.

Description:

This function synchronizes a widget’s extent with its position and size resources. Any region adjustments, blits, or damages are performed automatically.

This function calls the widget’s Extent method to calculate its new extent.

Returns:

- 0 No action.
- Nonzero The widget was modified or damage was generated.

Classification:

- Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
PtRemoveWidgetData()
Remove data from the widget data chain

Synopsis:

```c
int PtRemoveWidgetData(
    PtWidget_t *widget,
    PtWidgetClassRef_t *type,
    long subtype);
```

Arguments:

- `widget`: A pointer to the widget from whose chain you want to remove the data.
- `type`: The class of the widget removing the data.
- `subtype`: The subtype of data to remove. This is used to discriminate between multiple blocks of data added by the same widget class.

Description:

This function removes a piece of data from the widget data chain and frees the memory allocated for the data.

Returns:

0 if the data was found and released; -1 if it couldn’t be found.

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PtAddWidgetData(), PtFindWidgetData(), PtFindNextWidgetData()
PtResizeCanvas()
Set the size of a widget’s canvas

Synopsis:

```
int PtResizeCanvas( PtWidget_t *widget,
                   PhDim_t const *render );
```

Arguments:

- `widget` A pointer to the widget whose canvas you want to set.
- `render` A pointer to a `PhDim_t` structure (see the Photon Library Reference) that defines the desired size of the canvas.

Description:

This function changes the canvas belonging to the given widget to be the size specified in the `PhDim_t` pointed to by `render`.

This function is similar to `PtAttemptResize()`, but is easier to use.

The widget’s actual size, `widget->area.size`, is modified. If the widget’s resize policy prevents `PtResizeCanvas()` from adjusting the widget’s size (i.e. the canvas won’t fit within the provided render rectangle), `PtResizeCanvas()` sets the Pt_UCLIP bit in the widget’s resize flags. If this bit is set, the widget’s Draw method should apply clipping via `PtClipAdd()` before rendering its data.

Returns:

- 0 if the size didn’t change, or nonzero if the height or width changed.

Classification:

Photon

Safety

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

- `PtAttemptResize()`, `PtChildBoundingBox()`, `PtClipAdd()`, `PtClipRemove()`
- `PgExtentText()`, `PhDim_t`, `PtCalcCanvas()` in the Photon Library Reference
**Synopsis:**

```c
int PtResizePolicy( PtWidget_t *widget );
```

**Arguments:**

- `widget` A pointer to the widget to check for a resize policy.

**Description:**

This function determines whether a resize policy is currently in effect for the specified `widget`.

**Returns:**

- **0** No resize policy.
- **Nonzero** A resize policy is in effect. This returned value is a combination of the resize policy flags in effect.

**Examples:**

You can test the return of `PtResizePolicy()` to determine which policies are in effect:

```c
int resizePolicy;
resizePolicy=PtResizePolicy(myButton);

// test if any resize policies are set:
if(resizePolicy & Pt_RESIZE_XY_BITS == 0) {
    // no policies are set: do some stuff
}
```

**Classification:**

- **Photon**

**Safety**

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
PtSetExtentFromArea()

Calculate the extent of a widget

Synopsis:

```c
PhRect_t * PtSetExtentFromArea(
    PtWidget_t * widget,
    PhArea_t const * area,
    PhRect_t * extent);
```

Arguments:

- `widget` A pointer to the widget for which you want to calculate the extent.
- `area` A pointer to a `PhArea_t` structure that defines the area that the extent will be based on.
- `extent` A pointer to a `PhRect_t` where the function can store the extent.

Description:

This function calculates the extent rectangle of `widget` and places the result in the structure pointed to by `extent`. The extent is calculated based on the `area`, and the widget’s borders only (i.e. no anchoring or resize policy is enforced).

Returns:

The same pointer as the `extent` rectangle.

Classification:

Photon

Safety

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

`PhArea_t`, `PhRect_t` in the Photon `Library Reference`
PtSetStruct()
Set the specified resource

Synopsis:

```c
int PtSetStruct( char *base,
                 PtResourceRec_t const *mod,
                 PtArg_t const *arg );
```

Arguments:

- **base**: The base address of the widget whose resources are being modified.
- **mod**: A resource record that describes the type of resource being modified.
- **arg**: A pointer to a `PtArg_t` structure (see the Photon Library Reference) that indicates which resource to modify, as well as the value to assign to it.

Description:

This function sets the resource specified by `arg` on the widget at address `base` in the manner described by `mod`.

Unlike `PtSetValue()`, `PtSetStruct()` doesn’t examine the `mod_f` member of `mod` (e.g. for a Pt_CHANGE_RESIZE flag). It sets the appropriate structure members based on `mod->value` and `mod->len`.

If you want to do the extra widget processing as well as set the appropriate structure members, use `PtSetValue()`. For a cleaner solution, consider using `PtSuperClassSetResources()`.

Returns:

- 0: No change.
- 1: A change was made.

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

See also:

- `PtGetStruct()`, `PtSetValue()`, `PtSuperClassGetResources()`,
- `PtSuperClassSetResources()`
PtArg_t in the Photon Library Reference
PtSetValue()

Set the value of a resource using mod_f

Synopsis:

```c
int PtSetValue( PtWidget_t *widget,
                  PtResourceRec_t const *mod,
                  PtArg_t const *args );
```

Arguments:

- `widget` A pointer to the widget whose resource is being modified.
- `mod` A resource record that describes the type of resource being modified.
- `arg` A pointer to a `PtArg_t` structure (see the Photon Library Reference) that indicates which resource to modify, as well as the value to assign to it.

Description:

This function sets the value of a particular resource using the information provided in `mod`, including the `mod_f` member.

Consider calling `PtSuperClassSetResources()` for a cleaner solution. To set structure members without any further action, call `PtSetStruct()`.

Returns:

- 0 Nothing has changed.
- 1 The value has changed.
- 2 The value and length have changed.

Classification:

- **Photon**

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

- `PtSetStruct()`, `PtSuperClassSetResources()`
- `PtArg_t` in the Photon Library Reference
PtSuperClassCalcOpaque()

Call the Calc Opaque Rect method of specified superclass

Synopsis:

```c
void PtSuperClassCalcOpaque(
    PtWidgetClassRef_t *cref,
    PtWidget_t *widget);
```

Arguments:

- `cref` The superclass whose method you want to call.
- `widget` A pointer to the widget for which to call the superclass’s method.

Description:

This function calls the Calc Opaque Rect method of the specified superclass. Typically, this function calculates the rectangle representing the portion of the widget that isn’t transparent and sets or clears the widget’s Pt_OPAQUE flag.

The opaque rectangle for a widget is usually its canvas rectangle or the canvas rectangle derived from its PtBasic-class level.

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
Synopsis:

void PtSuperClassChildCreated(
    PtWidgetClassRef_t *cref,
    PtWidget_t *widget,
    PtWidget_t *child);

Arguments:

cref The container superclass whose method you want to call.

widget A pointer to the widget for which to call the superclass’s method.

child A pointer to the newly created widget.

Description:

This function invokes the Child Created method of the container class specified by cref.

Classification:

Photon

Safety

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
Invoke a Child Destroyed method

Synopsis:

```c
void PtSuperClassChildDestroyed(  
    PtWidgetClassRef_t *cref,  
    PtWidget_t *widget,  
    PtWidget_t *child);
```

Arguments:

- `cref` The container superclass whose method you want to call.
- `widget` A pointer to the widget for which to call the superclass’s method.
- `child` A pointer to the widget that’s just been destroyed.

Description:

This function invokes the Child Destroyed method of the container class specified by `cref`.

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
PtSuperClassChildGettingFocus()
Invoke a Child Getting Focus method

Synopsis:

```c
int PtSuperClassChildGettingFocus(
    PtWidgetClassRef_t *cref,
    PtWidget_t *widget,
    PtWidget_t *child,
    PhEvent_t *event);
```

Arguments:

- `cref` The container superclass whose method you want to call.
- `widget` A pointer to the widget whose child is getting focus.
- `child` A pointer to the widget that’s getting focus.
- `event` A pointer to a `PhEvent_t` structure (see the Photon Library Reference) that describes the event that’s causing the change in focus.

Description:

This function invokes the Child Getting Focus method of the container class specified by `cref`.

Returns:

- `Pt_END` The container prevented the child from having focus.
- `Pt_CONTINUE` The child may obtain focus.

Classification:

Photon

Safety

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

- `PhEvent_t` in the Photon Library Reference
Invoke a Child Getting Resources method

Synopsis:

```c
int PtSuperClassChildGettingResources(
    PtWidgetClassRef_t *cref,
    PtWidget_t *widget,
    PtWidget_t *child,
    PtArg_t *argt);
```

Arguments:

- `cref` The container class whose method you want to invoke.
- `widget` A pointer to the widget whose child is having one of its resources set.
- `child` A pointer to the widget that's having one of its resources set.
- `argt` A pointer to a `PtArg_t` structure (see the Photon Library Reference) that describes the resource being retrieved and where to store its value.

Description:

This function invokes the Child Getting Resources method of the container class specified by `cref`.

Returns:

- `Pt_END` The container prevented the resource query from receiving further processing — the container may have already satisfied the query.
- `Pt_CONTINUE` The resource query may be applied to the child.

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

`PtArg_t` in the Photon Library Reference
Synopsis:

```c
int PtSuperClassChildLosingFocus(
    PtWidgetClassRef_t *cref,
    PtWidget_t *widget,
    PtWidget_t *child,
    PhEvent_t *event
);
```

Arguments:

- `cref` - The container superclass whose method you want to call.
- `widget` - A pointer to the widget whose child is losing focus.
- `child` - A pointer to the widget that's losing focus.
- `event` - A pointer to a `PhEvent_t` structure (see the Photon Library Reference) that describes the event that's causing the change in focus.

Description:

This function invokes the Child Losing Focus method of the container class specified by `cref`.

Returns:

- `Pt_END` - The container prevented the child from losing focus.
- `Pt_CONTINUE` - The child may lose focus.

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

See also:

`PhEvent_t` in the Photon Library Reference
PtSuperClassChildMovedResized()

Invoke a Child Moved/Resized method

Synopsis:

```c
void PtSuperClassChildMovedResized(
    PtWidgetClassRef_t *cref,
    PtWidget_t *widget,
    PtWidget_t *child,
    PhArea_t *current_area,
    PhRect_t *current_extent,
    PhArea_t *old_area,
    PhRect_t *old_extent );
```

Arguments:

cref  The container class whose method you want to invoke.

widget A pointer to the widget whose child was resized.

child A pointer to the widget that was resized.

current_area A pointer to a PhArea_t structure that defines the widget’s current area (after the resize).

current_extent A pointer to a PhRect_t structure that defines the widget’s current extent (after the resize).

old_area A pointer to a PhArea_t structure that defines the widget’s old area (before the resize).

old_extent A pointer to a PhRect_t structure that defines the widget’s old extent (before the resize).

Description:

This function invokes the Child Moved Resized method of the container class specified by cref.

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
See also:

```
PhArea_t, PhRect_t
```
in the Photon *Library Reference*
PtSuperClassChildRealized()

Invoke a Child Realized method

Synopsis:

```c
void PtSuperClassChildRealized(
    PtWidgetClassRef_t *cref,
    PtWidget_t *widget,
    PtWidget_t *child);
```

Arguments:

- `cref` The container class whose method you want to invoke.
- `widget` A pointer to the widget whose Child Realized method is being invoked.
- `child` A pointer to the widget that has been realized.

Description:

This function invokes the Child Realized method of the container class specified by `cref`.

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
PtSuperClassChildSettingResources()

Invoke a Child Setting Resources method

Synopsis:

```c
int PtSuperClassChildSettingResources(
    PtWidgetClassRef_t *cref,
    PtWidget_t *widget,
    PtWidget_t *child,
    PtArg_t const *argt);
```

Arguments:

- `cref` The container class whose method you want to invoke.
- `widget` A pointer to the widget whose child is having one of its resources set.
- `child` A pointer to the widget that’s having one of its resources set.
- `argt` A pointer to a `PtArg_t` structure (see the Photon Library Reference) that describes the resource being set and its new value.

Description:

This function invokes the Child Setting Resources method of the container class specified by `cref`.

Returns:

- `Pt_END` The container prevented any further changes to the resource — the container may have already applied the change to the child.
- `Pt_CONTINUE` Further changes to the child’s resource may be applied.

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

`PtArg_t` in the Photon Library Reference
Invoke a Child Unrealized method

Synopsis:

```c
void PtSuperClassChildUnrealized(
    PtWidgetClassRef_t *cref,
    PtWidget_t *widget,
    PtWidget_t *child);
```

Arguments:

- `cref` The container class whose method you want to invoke.
- `widget` A pointer to the widget whose Child Unrealized method is being invoked.
- `child` A pointer to the widget that has just been unrealized.

Description:

This function invokes the Child Unrealized method of the container class specified by `cref`.

Classification:

- Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>
Synopsis:

```c
int PtSuperClassConnect(
    PtWidgetClassRef_t *cref,
    PtWidget_t *widget);

int PtSuperClassConnectFrom(
    PtWidgetClassRef_t *cref,
    PtWidget_t *widget);
```

Arguments:

- `cref` The superclass whose method you want to invoke.
- `widget` A pointer to the widget for which to invoke the method.

Description:

`PtSuperClassConnect()` invokes the Connection method of the specified widget class `cref`.

`PtSuperClassConnectFrom()` calls the Connection methods of all superclasses starting with the superclass specified by `cref`.

Returns:

- `Pt_CONTINU`E The connection callback chain can continue
- `Pt_HALT` or `Pt_END` The connection callbacks should stop.

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
PtSuperClassDraw()

Invoke the Draw method of the specified superclass

Synopsis:

```c
void PtSuperClassDraw( PtWidgetClassRef_t *wc_ref,
                       PtWidget_t *widget,
                       PhTile_t const *damage );
```

Arguments:

- `wc_ref` The superclass whose method you want to invoke.
- `widget` A pointer to the widget being drawn.
- `damage` A pointer to a list of PhTile_t structures (see the Photon Library Reference) that define the damaged areas to be drawn.

Description:

This function invokes the Draw method of the specified superclass `wc_ref`. Use this function to save code and complexity in your subclass’s Draw method.

Examples:

```c
static void my_draw( PtWidget_t *widget, PhTile_t *damage )
{
    // draw fill & borders as needed.
    PtSuperClassDraw( PtBasic, widget, damage );

    PtCalcCanvas( widget, &canvas );

    // Check if PtAttemptResize() function set Pt_UCLIP flag
    // in the Extent method.
    if( widget->resize_flags & Pt_UCLIP )
        PtClipAdd( widget, &canvas );

    PgDrawLine( canvas.ul, canvas.lr );
    PgDrawLine( canvas.lr, canvas.ul );

    if( widget->resize_flags & Pt_UCLIP )
        PtClipRemove();
}
```

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
See also:

`PhTile_t` in the Photon *Library Reference*
Invoke the Extent method of the specified superclass

Synopsis:

```c
void PtSuperClassExtent( PtWidgetClassRef_t *wc_ref,
                        PtWidget_t *widget );
```

Arguments:

- **wc_ref**: The superclass whose method you want to invoke.
- **widget**: A pointer to the widget for which to invoke the method.

Description:

This function invokes the Extent method of the specified superclass `wc_ref`.

The `PtLabel` class’s Extent method handles multiline text (with underlining and margins) and the resize policy.

The `PtContainer` class’s Extent method handles the resize policy for all its subclasses.

The `PtGraphic` class’s Extent method calculates an extent based on an array of points, line weight, origin, and resize policy.

Examples:

To have a widget honor a minimum x dimension resource:

```c
static void my_container_extent( PtWidget_t *widget )
{
    MyWidget_t *mw = (MyWidget_t *)widget;
    PhRect_t canvas, render, old_extent;
    old_extent = widget->extent;

    if( PtResizePolicy( widget ) )
    {
        PtChildBoundingBox( widget, &canvas, &render );
        if( render.lr.x - render.ul.x + 1 < mw->min_x )
            render.lr.x = render.ul.x + mw->min_x -1;
        PtAttemptResize( widget, &canvas, &render );
    }else if( widget->area.size.x < mw->min_x )
        widget->area.size.x = mw->min_x;

    PtSuperClassExtent( PtBasic, widget );

    // Apply anchoring.
    PtApplyAnchors( widget );

    if( memcmp( &widget->area, &old_area, sizeof( old_area ) ) )
        PtInvokeResizeCallbacks( widget );
}
```
Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
Synopsis:

```c
int PtSuperClassGetResources(
    PtWidgetClassRef_t *wc_ref,
    PtWidget_t *widget,
    int num_args,
    PtArg_t *args);
```

Arguments:

- `wc_ref` The superclass whose method you want to call.
- `widget` A pointer to the widget for which to call the superclass’s method.
- `num_args` The number of entries in the `args` array.
- `args` An array of `PtArg_t` structures (see the Photon Library Reference) that define which resources to get.

Description:

This function gets resources given by `num_args` and `args` in the manner defined by `wc_ref`. This is extremely helpful when overriding the `query_f` member of an overridden resource.

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

`PtArg_t` in the Photon Library Reference
Invoke the Got Focus method of the specified superclass

**Synopsis:**

```c
void PtSuperClassGotFocus( PtWidgetClassRef_t *cref,
                        PtWidget_t *widget,
                        PhEvent_t *event );
```

**Arguments:**

- `cref`        The superclass whose method you want to call.
- `widget`      A pointer to the widget for which to call the superclass’s method.
- `event`       A pointer to a `PhEvent_t` structure (see the Photon Library Reference) that describes the event.

**Description:**

This function invokes the Got Focus method of the superclass specified by `cref`. The `PtBasic` class’s Got Focus method is the method that normally invokes the widget’s Got Focus method or the user-level Got Focus callback.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

`PhEvent_t` in the Photon Library Reference
Invoke the Initialize method of specified widget class

Synopsis:

```c
int PtSuperClassInit( PtWidgetClassRef_t *cref,
                       PtWidget_t *widget);

int PtSuperClassInitFrom( PtWidgetClassRef_t *cref,
                           PtWidget_t *widget);
```

Arguments:

- `cref` The superclass whose method you want to call.
- `widget` A pointer to the widget for which to call the superclass’s method.

Description:

`PtSuperClassInit()` invokes the Initialize method of the specified widget class `cref`.

`PtSuperClassInitFrom()` calls the Initialize method of all superclasses starting with the superclass specified by `cref`. When `PtSuperClassInitFrom()` is used in a Initialize method, `Pt_END` should be returned.

Returns:

- `Pt_CONTINUE` The Initialization method chain can continue.
- `Pt_HALT` or `Pt_END` The Initialization method chaining should stop.

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
Synopsis:

```c
int PtSuperClassLostFocus(
    PtWidgetClassRef_t *cref,
    PtWidget_t *widget,
    PhEvent_t *event);
```

Arguments:

- `cref` The superclass whose method you want to call.
- `widget` A pointer to the widget for which to call the superclass’s method.
- `event` A pointer to a `PhEvent_t` structure (see the Photon Library Reference) that describes the event.

Description:

This function invokes the Lost Focus method of the superclass specified by `cref`.

The `PtBasic` class’s Lost Focus method is the one that normally invokes the widget’s Lost Focus method or the user-level Lost Focus callback.

Returns:

The maximum value returned by the Lost Focus callback chain.

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

`PhEvent_t` in the Photon Library Reference
Invoke the raw callback list of the specified widget class

Synopsis:

```c
int PtSuperClassRawEvent( PtWidgetClassRef_t *cref,
    PtWidget_t *widget,
    void *data,
    PtCallbackInfo_t *cbinfo );

int PtSuperClassRawEventFrom(
    PtWidgetClassRef_t *cref,
    PtWidget_t *widget,
    void *data,
    PtCallbackInfo_t *cbinfo );
```

Arguments:

cref  The superclass whose method you want to call.

widget  A pointer to the widget for which to call the superclass’s method.

data  Arbitrary data to pass to the raw callbacks as the client_data argument.

cbinfo  Callback information to pass to the raw callbacks as the cbinfo argument.

Description:

*PtSuperClassRawEvent()* invokes the raw callback list of the specified widget class.

*PtSuperClassRawEventFrom()* calls the raw callback lists of all superclasses starting with the superclass specified by cref. When *PtSuperClassRawEventFrom()* is used in a class’s raw callback, the callback should return Pt_END.

Returns:

Pt_END if the event was consumed, Pt_CONTINUE if it wasn’t.

Classification:

Photon

Safety

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PtCallbackInfo_t in the Photon Widget Reference
Synopsis:

```c
void PtSuperClassRealized( PtWidgetClassRef_t *cref,
                          PtWidget_t *widget );
```

Arguments:

- `cref` The superclass whose method you want to call.
- `widget` A pointer to the widget for which to call the superclass’s method.

Description:

This function invokes the Realization method of the specified widget class `cref`.

Classification:

- Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
PtSuperClassSetResources()
Set resources

Synopsis:

```c
int PtSuperClassSetResources(
    PtWidgetClassRef_t *wc_ref,
    PtWidget_t *widget,
    int num_args,
    PtArg_t const *args)
```

Arguments:

- `wc_ref` The superclass whose method you want to call.
- `widget` A pointer to the widget for which to call the superclass’s method.
- `num_args` The number of entries in the `args` array.
- `args` An array of `PtArg_t` structures (see the Photon Library Reference) that define the resources to set and their values.

Description:

This function sets the resources given by `num_args` and `args` in the manner defined by `wc_ref`. This is extremely helpful when overriding the `query_f` member of an overridden resource.

Returns:

The number of resources set.

Examples:

```c
my_modify_area( PtWidget_t *widget, PtArg_t *argt )
{
    MyWidget_t *mw = (MyWidget_t *)widget;
    mw->area_changed = Pt_TRUE;
    // Apply the resource the same way a superclass does.
    PtSuperClassSetResources(
        widget->class_rec->superclass->cref,
        widget, 1, argt );
    // Set the resource so that it’s treated
    // like a PtBasic class widget.
    PtSuperClassSetResources( PtBasic, widget, 1, argt );
}
```

Classification:

Photon
### Safety

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

`PtArg_t` in the Photon *Library Reference*
Tell the widget library about a change in visibility

Synopsis:

PtUpdateVisibility( PtWidget_t *widget,
                   PhRect_t *rect );

Arguments:

widget A pointer to a container widget whose visibility has changed.
rect A pointer to a PhRect_t structure that defines the area over which the visibility has changed.

Description:

This function tells the widget library that a change has occurred that may affect the visibility of widgets that intersect with the provided rect in the given container widget. This change may make the intersecting widgets completely obscured, completely unobscured, or partially obscured. This information is used to optimize refreshing the screen:

- Widgets that are completely obscured aren’t drawn (since you wouldn’t see them anyway).
- Widgets that are completely unobscured are drawn and don’t cause widgets above them to be repaired (a completely unobscured widget has no intersecting widgets above it).
- Otherwise a widget is drawn when damaged and causes all intersecting widgets above it to be repaired.

Returns:

0 on success; -1 if widget isn’t a container.

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
See also:

PhRect_t in the Photon Library Reference
Get the widget that’s above a given widget in a family hierarchy

Synopsis:

```c
PtWidget_t * PtWidgetAbove( PtWidget_t *root,
                           PtWidget_t *widget);
```

Arguments:

- **root**: A pointer to the highest widget that you want to consider in the widget family hierarchy.
- **widget**: A pointer to the widget for which you want to find a widget above.

Description:

This function returns a pointer to the widget that’s in front of `widget`, provided the returned widget isn’t any higher in the widget family hierarchy than `root`.

`PtWidgetAbove()` skips any disjoint widgets that it finds as it traverses the hierarchy.

Returns:

A pointer to the widget above, or NULL if there isn’t one. If `widget` is the same as `root`, the function returns the first child of `root`, or NULL if `root` has no children.

Classification:

Photon

Safety

- Interrupt handler: No
- Signal handler: No
- Thread: No

See also:

- `PtWidgetBrotherBehind()`, `PtWidgetBrotherInFront()`, `PtWidgetChildBack()`, `PtWidgetChildFront()`, `PtWidgetFamily()`, `PtWidgetParent()`, `PtWidgetTree()`, `PtWidgetTreeTraverse()` in the Photon Library Reference
This chapter describes the resource plugin API for creating your own resource editors for PhAB.

In PhAB, widgets have resources that are displayed in the Resources control panel. You modify these resources using either a small, specialized “frugal” editor directly in the control panel, or by opening a more complex “full” editor.

Previously the code for these resource editors was statically built into PhAB. Now the editors are resource editor plugins — they are small pieces of code and data put together into a DLL that PhAB loads at runtime. The purpose of this chapter is to describe how you can write your own resource editors.

Any resource has a type and a person that writes a resource editor plugin will write a plugin for one of these defined types:

- RES_DATATYPE_STRING
- RES_DATATYPE_MULTI
- RES_DATATYPE_CODE
- RES_DATATYPE_FONT
- RES_DATATYPE_COLOR
- RES_DATATYPE_DOUBLE
- RES_DATATYPE_NUMERIC
- RES_DATATYPE_CHOICE
- RES_DATATYPE_FLAG
- RES_DATATYPE_PATTERN
- RES_DATATYPE_LIST
- RES_DATATYPE_PIXMAP
- RES_DATATYPE_ALLOC

Each of these types is described in the section Resource datatypes. If you write a custom widget and devise a new resource that doesn’t seem to fit in any of the above types, then you should use RES_DATATYPE_ALLOC since it describes a general resource datatype. This resource type consists of a buffer of bytes, with a given length. A protocol for ensuring the endian safety when saving/loading the alloc resource is documented in the section Endian issues.

Once you’ve chosen one of these types for a newly devised resource, the resource type is recorded in the widget’s palette file as a string. See the sections Editing palette.def and Creating a widget description table in the Binding Widgets into PhAB chapter.

You can put one, two or more editors into the same DLL, or you can put a frugal editor into one DLL and a full editor into a different DLL. You can choose to only write the
full editor and let the frugal editor be the predefined one, or use the frugal editor written by another user, and so on.

To decide which resource editor should be used for a resource type, PhAB looks at the `res_editors.def` to match entries with the `abstract_name_of_editor` specified in the `palette.def` file. There is a `res_editors.def` file for each user on a system, and a global `res_editors.def` file for all users. See The `res_editors.def` file section for more information about the structure of this file.

With the resource editor plugin API, you can:

- override the predefined resource editors. For instance, you might want to override the `RES_DATATYPE_PIXMAP` resource editor with your own. You specify the new editor in the `palette.def` file.

- write a specialized resource editor for a datatype that already exists.
  For instance, if you have a new widget class that has a new resource representing a date (`time_t`), the default numeric resource editor might be inappropriate. Instead, you can specify that for this date resource, an alternate custom resource editor is used.

- devise a brand new resource type using the `RES_DATATYPE_ALLOC` type and write a resource editor for it. PhAB and the aplib will know how to save it and load it at runtime, and the resource editor plugin gives you a way to change it in PhAB.

This chapter contains:

- Resource datatypes — describes the resource datatypes supported by the plugin API.
- The life cycle of a plugin instance — describes the life cycle of of resource editor plugins.
- The `res_editors.def` file — describes the configuration file you need to edit when you create a new resource and resource editor.
- Endian issues — describes how endian issues are handled with the `RES_DATATYPE_ALLOC` type.

See these chapters for more information about building a resource editor using the plugin API:

- The resource plugin API
- An example resource editor plugin

**Resource datatypes**

When you create a new resource, for example for a new widget, you need to tell PhAB about the resource and how it should be handled. You need to enroll the resource in one of the types listed below by entering the resource in the widget’s palette file. For
more information on this, see Creating a widget description table in the Binding Widgets into PhAB chapter.

A widget resource can be one of the following datatypes:

- **RES_DATATYPE_CHOICE** — an exclusive choice resource type, for example
  \texttt{Pt\_ARG\_LABEL\_TYPE}.

- **RES_DATATYPE_CODE** — a function resource type, for example
  \texttt{Pt\_ARG\_RAW\_DRAW\_F}.

- **RES_DATATYPE_COLOR** — a color resource type, for example
  \texttt{Pt\_ARG\_FILL\_COLOR}.

- **RES_DATATYPE_DOUBLE** — a numeric double resource type, for example
  \texttt{Pt\_ARG\_NUMERIC\_VALUE} for a \texttt{PtNumericFloat}.

- **RES_DATATYPE_FLAG** — a flag resource type, for example \texttt{Pt\_ARG\_FLAGS}.

- **RES_DATATYPE_FONT** — a font resource type, for example
  \texttt{Pt\_ARG\_TEXT\_FONT}.

- **RES_DATATYPE_LIST** — a list resource type, for example \texttt{Pt\_ARG\_ITEMS}.

- **RES_DATATYPE_MULTI** — a multi line string resource type, for example
  \texttt{Pt\_ARG\_TEXT\_STRING}.

- **RES_DATATYPE_NUMERIC** — a numeric resource type, for example
  \texttt{Pt\_ARG\_NUMERIC\_VALUE} for a \texttt{PtNumericInteger}.

- **RES_DATATYPE_PATTERN** — a pattern resource type, \texttt{Pt\_ARG\_FILL\_PATTERN}.

- **RES_DATATYPE_PIXMAP** — an image resource type, for example
  \texttt{Pt\_ARG\_LABEL\_IMAGE}.

- **RES_DATATYPE_STRING** — a string resource type, for example
  \texttt{Pt\_ARG\_WINDOW\_TITLE}.

- **RES_DATATYPE_ALLOC** — a generic resource type, not covered by the other types.
  For example, \texttt{Pt\_ARG\_RAW\_LAYOUT\_DATA}.

**Resource datatypes general notes**

Resource data is passed from PhAB to the plugin instance via a call to
\texttt{ResPluginSetDataF\_t}, and passed from the plugin instance to PhAB with \texttt{apply()},
\texttt{ResPluginAnyChangesF\_t}, or \texttt{answer\_changes()}. Each of these functions has a
\texttt{value} and an \texttt{n} parameter for the data, though for each resource type these parameters
are used differently.

For some resource datatypes, additional format information is required. Format data is
passed as a pointer to this structure:
typedef struct ResPluginFormatData_ {
  int n;
  const void *value;
} ResPluginFormatData_t;

Each resource data type uses the members of this structure differently. The resource editor should not assume that the pointer to `ResPluginFormatData_t` remains valid beyond the call that passed the pointer.

For some datatypes, the “master copy rule” applies, which means PhAB maintains a master copy of the data and these conditions apply:

- When this data is being passed from PhAB to the plugin, PhAB holds the “master copy” and the value is guaranteed to remain a valid pointer until PhAB informs the plugin about a new value by calling the `ResPluginSetDataF_t` method. When the `ResPluginSetDataF_t` method is called, the old master copy is still valid.

- When this data is passed from the plugin to PhAB, PhAB frees its master copy and the data passed by the plugin becomes the master copy.

For other datatypes, the master copy rule doesn’t apply. In these cases, when the data is passed, the recipient (either the plugin or PhAB) must make its own copy of the passed data.

RES_DATATYPE_ALLOC

This is the generic datatype that you can use for newly devised resources that do not fit any of the other predefined types. There is no predefined resource editor for this data type. An example of this resource type is `Pt_ARG_ROW_LAYOUT_DATA`.

When this datatype is passed between the plugin and PhAB, in the passing function the parameter `value` is an array of bytes and `n` is the number of bytes being passed.

To allocate memory for this data type in your resource editor, you should use the `alloc_mem()` function exported by PhAB to allocate memory, and `free_mem()` to free it.

The palette file `type` string for this resource type is `alloc`.

This resource datatype does not require additional format information from `ResPluginFormatData_t`.

RES_DATATYPE_CHOICE

This data type represents a list of choices, which are string and numeric value pairs that a user can choose from. An example of this resource type is `Pt_ARG_LABEL_TYPE`.

When this datatype is passed between the plugin and PhAB, in the passing function the parameter `value` is not used. The `n` parameter contains the numeric value.

The master copy rule doesn’t apply, so the plugin should make its own copy of the data when it is passed.
The `ResPluginFormatData_t` structure is used to indicate the list of choices and the associated numeric values. In the structure, `value` is an array of `ResPluginChoiceFormat_t*`, and `n` indicates the number of the items in the array.

The resource editor can safely assume that the array will remain valid beyond the call that passed this structure (that is, it will stay valid during the existence of the plugin).

The `ResPluginChoiceFormat_t` is defined as:

```c
typedef struct {
    char *name;
    long value;
} ResPluginChoiceFormat_t;
```

The members are:

- `name` the choice name
- `value` the choice value

The value being passed to the plugin or from the plugin has to be one of the values indicated in the format structure.

The palette file type string for this resource type is `choice`.

**RES_DATATYPE_CODE**

This data type represents a function, for example `Pt_ARG_RAW_DRAW_F`. It is similar to the `RES_DATATYPE_STRING` resource type, except that the master copy rule doesn’t apply. This means that you have to make a copy of the value in the resource editor plugin, because it might point to memory on the stack, which is invalid beyond the call that passed the value.

An editor for this resource should allow the user to edit one line of characters.

The resulting line of characters should be a valid string for specifying a function name in PhAB. For more information on properly formatting this string, see Function names and filenames in the Working with Code chapter of the Photon Programmer’s Guide.

The `ResPluginFormatData_t` structure is used to indicate the prototype of the function. The `n` member is not used and the `value` member is a pointer to the following structure:

```c
/* format for code datatype */
typedef struct {
    char *prototype;
} ResPluginCodeFormat_t;
```

The `prototype` member should point to a function prototype string. For example:

```c
PtWidget_t* @( PtWidget_t *window, PtWidget_t *widget, int position, char *text, char *font, PgColor_t fill_color, PgColor_t text_color )
```
The @ is used to indicate the function name place holder. The plugin could use the provided prototype to display a list of already defined functions that match the prototype.

The resource editor can safely assume that the value pointer in the format structure will remain valid beyond the call that passed this structure. In fact, it will stay valid during the existence of the plugin.

The palette file type string for this resource type is code.

**RES_DATATYPE_COLOR**

This data type represents a color resource, for example Pt_ARG_FILL_COLOR.

When this datatype is passed between the plugin and PhAB, in the passing function the value parameter is not used. The n parameter contains the color, which is a PgColor_t type.

The master copy rule doesn’t apply, so your plugin should make a copy of this data when it receives it.

No format data is required.

The palette file type string for this resource type is crgb.

**RES_DATATYPE_DOUBLE**

This data type represents a double-precision floating point value, such as Pt_ARG_NUMERIC_VALUE for PtNumericFloat.

When this datatype is passed between the plugin and PhAB, in the passing function the value parameter is a pointer to a double representing the value, and the n parameter is not used.

The master copy rule applies.

The ResPluginFormatData_t structure indicates the range of values the resource editor should allow this resource to have. In ResPluginFormatData_t, the n member is not used and the value member is a pointer to this structure:

```c
typedef struct {
    double min;
    double max;
} ResPluginDoubleFormat_t;
```

The min and max members indicate the minimum and maximum values allowed for the data type. The resource editor can safely assume that the value pointer in the format structure will remain valid beyond the call that passed this information. In fact, it will remain valid during the existence of the plugin.

The palette file entry for this data type is double. The palette file string for this resource type indicates the range this way:

```plaintext
r=manifest_name,desc_name,resource_no,reserved,
   select_ind,type, default_value[,additional_info]
```
where:

- `type` is `double`
- `additional_info` (if required) can be either:
  - `f` — specifies single-precision 32 bit floating point resources
  - `min/max` — specifies the explicit minimum and maximum values

For example:

\[
r=Pt_{-}ARG_{-}NUMERIC_{-}INCREMENT,Numeric\;increment,53003,\;0,0,\;double,1,0/100
\]

Also see the chapter Binding Widgets into PhAB in the *Photon Programmer’s Guide*.

**RES_DATATYPE_FLAG**

This data type represents a flag resource, for example `Pt_{-}ARG_{-}FLAGS`.

When this datatype is passed between the plugin and PhAB, in the passing function the `value` parameter is an unsigned long integer representing the value, and the `n` parameter is not used.

The master copy rule doesn’t apply.

The `ResPluginFormatData_t` structure is used to indicate the list of values, names and masks. In the `ResPluginFormatData_t` structure, the `value` member is an array of `ResPluginFlagFormat_t*`, and the `n` member is the number of items in the array.

The resource editor can safely assume that the array of `ResPluginFlagFormat_t*` will remain valid beyond the call that passed this structure. In fact, it will stay valid during the existence of the plugin.

The `ResPluginFlagFormat_t` is defined as:

```c
typedef struct {
    char *name;
    unsigned long value;
    unsigned long mask;
} ResPluginFlagFormat_t;
```

where `name` is the flag’s name, `value` is the flag’s value and `mask` is the associated mask.

The flags should be presented as names in the resource editor, so that a user can choose one or more values. When the user chooses more than one value, the resulting value is all selected values ORed together.

The `mask` controls how the different values interact with each other. All the values that match a bit in a mask are exclusive values, that is, they exclude each other. For instance if the current value of the resource is `current_value` and the user selects another flag of the `value` that has a `mask` associated, the resulting value will be:

\[
(current\_value \& (~ mask)) | value
\]
That means that first the mask is used to clear all the corresponding bits from the current_value and then the value is OR-ed in.

The mask can be 0, which means that the respective flag value doesn’t interact with any other flag values.

The palette file type string for this resource type is flag.

**RES_DATATYPE_FONT**

This data type represents a font resource, such as Pt_ARG_TEXT_FONT.

This type is the same as the RES_DATATYPE_STRING resource type, but the resulting string has to be a valid font name. For more information on generating a valid font name, see PgGenerateFontName().

The ResPluginFormatData_t structure is used to pass extra information about the fonts allowed for selection. The value member is not used, and the n member can be:

RES_DATATYPE_FONT_FIXED

This resource requires a fixed font.

To define a font resource in a palette file, use this string format:

```
  r=manifest_name,desc_name,resource_no,reserved,
       select_ind, type,default_value[,additional_info]
```

where type is font and if the font is fixed-width, additional_info is fixed.

**RES_DATATYPE_LIST**

This resource type represents a list, such as Pg_ARG_ITEMS.

When this datatype is passed between the plugin and PhAB, in the passing function the value parameter is an array of strings, the last item of the array being NULL, and the n parameter is the number of strings in the array, not including the last item.

The master copy rule applies.

No format data is required.

You should should use alloc_mem() exported from PhAB in your resource editor to allocate memory for this resource, and free_mem() to free it.

The palette file type string for this resource type is list.

**RES_DATATYPE_MULTI**

This resource type represents a multi-line text string, such as Pt_ARG_TEXT_STRING.

This resource type is the same as the RES_DATATYPE_STRING, with the difference that the RES_DATATYPE_MULTI type can contain newline \n characters. The resource editor should allow you to expand on multiple lines.

No format data is required.

The palette file type string for this resource type is multi.
**RES_DATATYPE_NUMERIC**

This resource type represents a numeric value, including short, integer, long signed and unsigned values. An example is `Pt_ARG_NUMERIC_VALUE`.

When this datatype is passed between the plugin and PhAB, in the passing function the `value` parameter is not used. The parameter `n` contains the numeric value.

The master copy rule doesn’t apply.

The `ResPluginFormatData_t` structure is used to indicate the range of values the editor should allow. In the structure, the `n` parameter is not used and the `value` parameter is a pointer to this structure:

```c
typedef struct {
   int64_t min;
   int64_t max;
} ResPluginNumericFormat_t;
```

The `min` and `max` fields indicate the minimum and maximum values allowed in the editor. The resource editor can safely assume that the `value` pointer in the format structure will remain valid beyond the call that passed this structure. In fact, it will remain valid during the existence of the plugin.

The palette file `type` string for this resource type is `numeric`. The previous string `short` is deprecated, but it is still recognized for backward compatibility. The palette file string for this resource type indicates the range this way:

```
r=manifest_name,desc_name,resource_no,reserved,
   select_ind,type, default_value[,additional_info]
```

where `type` is `numeric` and `additional_info` can be:

- `s8` — signed byte
- `s16` — signed short
- `s32` — signed int
- `u8`, `u16`, `u32` — the unsigned counterparts
- `s` — a shortcut for signed int
- `u` — a shortcut for unsigned int
- `min/max` the explicit minimum and maximum values

Also see the chapter Binding Widgets into PhAB.

If the `additional_info` section is missing, the resources is assumed to be `s32`, a signed int.
RES_DATATYPE_PATTERN

This data type represents a pattern.

When this datatype is passed between the plugin and PhAB, in the passing function the value parameter is a pointer to a PgPattern_t, and the n parameter is not used.

The master copy rule doesn’t apply, so the value pointer might be allocated on the stack and your plugin should make a copy of the data as soon as it receives the data.

No format data is required.

The palette file type string for this resource type is pattern.

RES_DATATYPE_PIXMAP

This data type represents an image map, for example Pt_ARG_LABEL_IMAGE.

When this datatype is passed between the plugin and PhAB, in the passing function the value parameter is a PhImage_t pointer and the n parameter is not used.

The master copy rule applies over the whole image. When this value is passed, the PhImage_t* is passed together with all associated data such as the image data and image palette.

No format data is required.

You should use alloc_image() exported from PhAB to allocate the image pointer, and free_image() to free it.

The palette file type string for this resource type is pixmap.

RES_DATATYPE_STRING

This data type represents a string.

When this datatype is passed between the plugin and PhAB, in the passing function the value parameter is an array of characters, NULL terminated, and the n parameter is set to the length of the string.

The master copy rule applies.

You should use the exported PhABExportAllocMemF_t to allocate memory, and PhABExportFreeMemF_t to free the memory.

This is the same as the RES_DATATYPE_MULTI type, with the difference that the data cannot contain ‘\n’ characters, and the resource editor should allow only one line of characters to be edited.

No format data is required.

The palette file type string for this resource type is string.
The life cycle of a plugin instance

The life cycle of a plugin depends on how it is written. A resource editor plugin can be a frugal editor, or a full editor, and the full editor can be implemented to use an external application. Each scenario is described here.

The frugal resource editor

The following scenario applies to a frugal resource editor:

- PhAB creates an instance of the plugin using the `ResPluginFrugalCreateF_t` method.
- The user interacts with the plugin
- The plugin can call into PhAB with the `apply()` method, requesting PhAB apply the current value.
- The plugin can call into PhAB with the `open()` method, requesting PhAB open a full editor to edit this resource.
- PhAB gives the plugin new data it just got from the full editor; the plugin must take the new data and throw away any modified version of the old data it might have.
- ...
- PhAB calls the `ResPluginDestroyF_t` method to destroy the plugin instance.

The full resource editor

The following scenario applies for a full resource editor that doesn’t use an external application:

- PhAB creates an instance of the plugin using its `ResPluginFullCreateF_t` method.
  In this method:
  - We recommend that in order to ensure a consistent look and feel for all resource editors, that you use the convenience function `create_window()`, exported from PhAB.
  - The `create_window()` function creates the resource editor window and fills in a `PtWidget_t**` with a pointer to the container widget that the plugin has to populate.
  - The plugin populates the container widget.
  - The plugin returns from its “create” method.
- Instead of using the `create_window()` convenience function, you can choose construct the resource window in the plugin yourself.
- The user edits the data inside the editor window.
The plugin should show its state graphically. If the window was created using `create_window()` it can use the convenience function `set_state()` to render the state.

The plugin can call into PhAB with the `apply()` method, requesting PhAB to apply the current resource editor value.

PhAB can call the `ResPluginAnyChangesF_t` method and the plugin should answer if there are changes to its data. The call to this function is made because PhAB wants to close or disable the editor’s window, or put a new value into the editor, and it wants to know what to do with the old value (for instance, when the selection changes). If there are changes, they are passed in this function call.

PhAB can call the `ResPluginDisableF_t` method, and the plugin has to throw away it’s value, and disable the content of the window, except for the Cancel button. If you used the `create_window()` function to create the resource editor window, use the `set_state()` function exported by PhAB to disable that part of the interface that was not created directly by you.

PhAB can later call `ResPluginSetDataF_t` and the plugin should re-enable itself and display the new data.

If the frugal editor asks PhAB to change the value while the full editor has some modification in it, then the full editor plugin will get a call to `ResPluginSetDataF_t` and the plugin is already enabled. The full editor can have one of the following behaviors:

1. Throw away its own modifications and use the new data passed from PhAB.
2. Keep the modifications it already has.

The user may press Cancel or close the editor’s window. In this case if the window was not created with `create_window()`, the plugin has to call into PhAB with the `closing()` method. If `create_window()` was used, then PhAB drives the plugin and the closing method shouldn’t be called.

PhAB calls `ResPluginDestroyF_t` to destroy the plugin instance.

**An external application resource editor**

When your full resource editor uses an external application, PhAB assumes that the plugin does not have access to the external application code — the external application is considered to be a black box. The only assumptions made are:

- The plugin can spawn the external application.
- The plugin has a way of passing data to the external application, for example, by putting the data into a file and giving the file to the external application.
- The plugin has a way of detecting when the external application was closed, for example by polling the PID or by attaching a SIGCHLD handler.
- The plugin has a way of detecting when the user has saved the changes, for example by using `stat()` to examine the modification time of a data file.
The plugin can send a Ph_WM_CLOSE event, and the external application displays a standard “save, discard, or cancel” dialog, before closing.

For an example of a resource editor that uses an external application, see the external application example in the Plugin Examples chapter. That example shows how you can use ped as an editor for a PtButton’s Pt_ARG_TEXT_STRING resource.

The following scenario applies for a full resource editor that uses an external application:

- PhAB creates an instance of the plugin using its ResPluginFullCreateF_t method.

- Inside this method, the plugin packages the resource data passed as an argument and gives it to the external application. For example, it might package the data into a file and spawn an external application, passing the file as an argument.

- The user starts editing the data inside the editor’s window.

- The plugin has a way to monitor the external application for the following events:
  - The user chooses Save in the external application. At this point, the plugin calls into PhAB with the apply() method.
  - The user closes the external application. At this point the plugin checks if there are changes and possibly calls apply, and then calls the exported closing() method. Because the user has already made the decision about saving or not saving the data, when PhAB calls the ResPluginAnyChangesF_t method, the plugin should return RESPLUGIN_NO_CHANGES inside the closing() method.

- PhAB can decide to call the ResPluginAnyChangesF_t method and the plugin should answer if there are changes or not. The call to this function is made because PhAB wants to close the editor’s window or disable the editor’s window, or PhAB wants to put a new value in the resource editor’s window and wants to know what to do with the old value (for instance, when the selection changes).

- If the plugin is able, it can answer right away whether there are changes or not. But most likely the plugin will not know and will want to let the external application decide if there are changes or not. Therefore, the plugin should send a Ph_WM_CLOSE event to the external application’s window and bring it to front, and then return RESPLUGIN_WAIT to the call to ResPluginAnyChangesF_t.

- At this point PhAB goes into a waiting state, in which the user cannot change the current selection and PhAB displays a message telling the user what’s happening.

- The external application will display a typical “save, discard, or cancel” dialog.

- Any choice other than Cancel will terminate the external application, the plugin will detect this and will call into PhAB with the answer_changes() method, giving PhAB the new data, if necessary. Note that at this point PhAB doesn’t mind if the plugin always replies that there are changes — it might be easier for the plugin to always grab the file and pass the data to PhAB).
• If the user chooses **Cancel**, then PhAB will remain in the waiting state until the user closes the external application and the plugin calls with the `answer_changes()` method. PhAB always assumes that a call to `answer_changes()` means the user has closed the editor’s window. The plugin doesn’t have to call the exported `closing()` method explicitly.

• PhAB calls `ResPluginDestroyF_t` to destroy the plugin instance.

The **res_editors.def** file

The mapping between the `abstract_name_of_editor` in the palette files and the actual frugal and full editors is done in the **res_editors.def** file (see Creating a widget description table in the Binding Widgets into PhAB chapter).

There are two **res_editors.def** files, one at the user level file located under the user’s home directory along with the other PhAB configuration files (`~/ph/phab`), and a global file, located in the same directory as the PhAB executable. The user’s **res_editors.def** file overrides the settings in the global file.

The format of both files is:

```plaintext
e=abstract_name_of_editor
[p=path]
[F=full_editor]
[p=path]
[f=frugal_editor]
```

Each resource definition in the palette files should have a corresponding `abstract_name_of_editor` definition (the `r=` line).

The lines that define resources in the palette file are supposed to have an `abstract_name_of_editor` for each resource definition line (the lines that begin with `r=`). See Creating a widget description table in the Binding Widgets into PhAB chapter. When the `abstract_name_of_editor` for a resource is missing, it is assumed that the `abstract_name_of_editor` is the datatype string.

For instance, for the `Pt_ARG_CONTAINER_FLAGS` definition, the `abstract_name_of_editor` and the datatype string are both `flag`:

```plaintext
r=Pt_ARG_CONTAINER_FLAGS,Container Flags,10005,10005,1,flag,0x30,9
```

Each one of the `abstract_name_of_editors` specified in all the palette files must have a section in at least one of the **res_editors.def** files (either the user specific or the global file). This section indicates the path and the name of the DLL that contains the code for the plugin. Please note that the predefined editors are statically built into PhAB and are not specified in the **res_editors.def** files. You can override these predefined editors by adding sections to the **res_editors.def** file.

For instance you can define a section `e=string` to override the default resource editor for the **RES_DATATYPE_STRING** resource datatype.

The **res_editors.def** file defines these settings:
e=abstract_name_of_editor

The settings for a particular resource editor. This line must match the abstract name of editor in a palette file.

p=path An optional setting, which can be used to specify the path to the next shared objects. It applies to only the shared objects specified inside the current e=abstract_name_of_editor section.

An empty line p= means the path is PhAB’s standard location for the plugin editors, which is plugins/resource in PhAB’s directory.

If the part following p= doesn’t start with a slash /, the path is considered to be relative to the user’s home directory.

If no path information is available when the F= or f= settings are parsed, then the path is considered to be PhAB’s standard location for the plugin editors, which is plugins/resource in PhAB’s directory.

If the setting is p=\$, then the $LD_LIBRARY_PATH environment variable is used.

F=full_editor, f=frugal_editor

Specifies the full and frugal editors. These lines have the form editor@dll_name.

The dll_name includes the shared library extension. On QNX the extension for shared objects is .so, while on Windows it is .so or .dll.

For example, the frugal_editor can be specified as: date@editors.so. If the @dll_name part is missing, the dll name is considered to be the same as the editor’s name plus the extension.

For instance, assume you want to write a resource editor to edit a date. The datatype used is numeric, but the default editors for the numeric datatype are not suitable for editing a date format:

a)

e=date
p=/tmp/plugins
F=date1@exp_date
p=
f=date2@date

In this example, the date type uses the full editor date1 located in the /tmp/plugins/exp_date.so, and the frugal editor date2, located in date.so, in PhAB’s standard plugin directory plugins/resource under the PhAB’s executable directory.

b)

e=date
f=date2
This example uses the date2 editor for the frugal editor, located in plugins/resource/date2.so.

c)

e=date
p=my_plugins
F=date

This example uses the full editor date located in my_plugins/date.so, under the user’s home directory.

If a palette file specifies an abstract name of an editor and there is no information about it in any of the res_editors.def files, and PhAB detects this at startup, it will exit with an error, except for the cases where it is safe to use the default datatype editor instead.

The def_res.def file in PhAB’s executable directory lists the abstract editors (one editor per line) for which it is safe to use the default datatype editor, anytime something prevents a different editor from being used.

For instance, in the example c) above, the frugal editor for date is not specified (there is no “f=” entry). If the predefined frugal editor for the datatype numeric is to be used, def_res.def has to specify date inside.

The RES_DATATYPE_ALLOC datatype can’t be listed in def_res.def since there is no default editor for it.

**Endian issues**

Because you can run PhAB on one platform and target another platform with a different endianness, PhAB needs to know how to load and save the widget files and the aplib needs to know how to load the content of the resource file at run time.

For the predefined datatypes, this is already taken care of, because the widget files are saved in an endian-insensitive form, and at runtime they are converted to the right endian form.

For the RES_DATATYPE_ALLOC datatype, you can specify an endian signature. This is a string that you put in the palette files. Each time a resource is saved, the string tells PhAB how to save the resource data in a endian-insensitive form. The string is saved along the resource data in the widget files. When the widget files are loaded either at runtime by aplib, or by PhAB, the endian string is read before reading the actual data. The string tells aplib and PhAB how to convert the loaded data to the right endian form.

Here’s an example of endian string:

24cs+il4c

This string describes an endian structure with:

- 24 characters (24 bytes endian insensitive)
• 2 shorts followed by an array of structures containing:
  - one integer (4 bytes)
  - one long (8 bytes)
  - 4 characters

Use:
• c for a character (1 byte endian insensitive)
• s for a "short" (2 bytes)
• i for an integer (4 bytes)
• l for a long (8 bytes)

Each of these can be preceded by a number to indicate a sequence of the same type data.

Use a + to indicate the start of an array of structures. The number of items in the array is determined by looking at the length of the resource data that’s left to be loaded for that particular resource.

You can only specify an endian string for RES_DATATYPE_ALLOC resource types, and it’s optional. A RES_DATATYPE_ALLOC resource type that has no endian string will be considered to be endian insensitive and in this case it will be the widget’s and the plugin’s responsibilities to correctly interpret the array of bytes.

In other words, when the endian string is missing, it’s the same as setting the endian string to +c.
Chapter 10

The resource editor API
This chapter lists the structures that support the resource editor plugin API, the functions exported by PhAB that support the API, and the functions you need to provide in your plugin.

Resource editor code is put in a DLL. You can put one, two, or more resource editors into a single DLL. Alternatively, you can put the frugal resource editor for a resource type in one DLL, and the full editor in another. Each DLL is organized as an addon that uses the Addon Interface API.

The DLL must declare the addon interface as:

```
#define RESPLUGIN_VERSION 202

AOInterface_t interfaces[] = 
{ 
    { "PhAB Resource Editors", RESPLUGIN_VERSION, &tab },
    { 0,0,0 }
};
```

Note that the RESPLUGIN_VERSION is defined as 202.

The `tab` variable must point to an instance of the `ResPlugin_t` structure.

**Resource Editor API structures**

You have to provide all the methods defined in `ResPlugin_t`, `ResPluginFrugalEditor_t`, and `ResPluginFullEditor_t` in the resource editor plugin, otherwise PhAB will not load and use the plugin.

When a new plugin instance is created, it is given a pointer to a structure that is exported from PhAB to the plugin and contains functions that the plugin can call into PhAB. All these functions take at least a `PhABHandle_t` as an argument.

Functions defined in `ResPlugin_t` that you must provide in the plugin:

- `ResPluginLoadingF_t` — a function called just after the DLL is loaded by PhAB into its address space, used to set up any resources common to the plugins in the DLL.

- `ResPluginUnloadingF_t` — a function called before the DLL is unloaded from PhAB’s address space. This function is called once for each successful call to `ResPluginLoadingF_t`.

Functions defined in `ResPluginCommon_t` that you must provide in the plugin:

- `ResPluginSetDataF_t` — a function to set the data value in a resource editor.

- `ResPluginDestroyF_t` — a function to destroy a resource editor and free any resources allocated.

Functions defined in `ResPluginFrugalEditor_t` that you must provide in the plugin:

- `ResPluginFrugalCreateF_t` — a function to create an instance of a frugal resource editor.
Functions defined in ResPluginFullEditor_t that you must provide in the plugin:

- **ResPluginAnyChangesF_t** — a function called by PhAB to determine whether the resource editor contains changes.
- **ResPluginBlockF_t** — a function called by PhAB to tell the resource editor to temporarily block input.
- **ResPluginDisableF_t** — a function called by PhAB to tell the resource editor to disable itself, and appear disabled (greyed out).
- **ResPluginFullCreateF_t** — a function to create an instance of a full resource editor.
- **ResPluginGetChangesF_t** — a function called by PhAB to retrieve any changes in the resource editor.
- **ResPluginGetAreaF_t** — a function called by PhAB to find out the area of the resource editor window.
- **ResPluginToFrontF_t** — a function called by PhAB to bring the resource editor window to the front.

Functions exported from PhAB

- **alloc_image()** — called by a plugin to allocate memory for RES_DATATYPE_PIXMAP data.
- **alloc_mem()** — called by a plugin to allocate memory for its copy of data.
- **answer_changes()** — called by a plugin to answer PhAB’s call to ResPluginAnyChangesF_t.
- **apply()** — called by a plugin to apply its data to the widget in PhAB.
- **closing()** — called by a plugin to tell PhAB that the resource editor will close (for example, the user clicked the Close button.)
- **cmp()** — a function a plugin can use to compare two ResDatatype_t values.
- **dup()** — a function a plugin can use to duplicate a ResDatatype_t value.
- **free()** — a function a plugin can use to free a ResDatatype_t value.
- **free_image()** — a function a plugin can use to free RES_DATATYPE_PIXMAP data allocated by alloc_image().
- **free_mem()** — a function a plugin can use to free memory allocated by alloc_mem().
- **get_res()** — a function to get resource data from another resource editor plugin, allowing one editor to use another as a “sub-editor”.
- **open()** — a function a frugal resource editor can call to open a full resource editor for its datatype.
The resource editor’s window supporting functions

These functions are exported from PhAB in the PhABResExportFull_t structure, which is passed as a pointer when the plugin instance is created. They are convenience functions that a full editor can use in order to ensure a consistent look for the resource editor windows. We recommend that you use these functions to create and manage the editor window, except for the case when your plugin spawns an external application. In that case the external application should provide the window.

- **create_window()** — a function that creates a window container which the plugin can use as the full resource editor window.
- **destroy()** — a function to destroy the window created by create_window().
- **get_area()** — a function to get the area of the window created by create_window().
- **message()** — a function that displays a message in PhAB.
- **set_state()** — a function to set the state of the window created by create_window().
- **to_front()** — a function to bring the window created by create_window() to the front.
PhABResExportCommon_t.alloc_image() © 2007, QNX Software Systems GmbH & Co. KG.

A function exported by PhAB to allocate memory for an image

**Synopsis:**

```c
PhImage_t* alloc_image ( PhImage_t *src );
```

**Arguments:**

`src` The PhImage_t image for which you want to allocate memory.

**Description:**

This function is exported from PhAB in the PhABResExportCommon_t structure.

This function allocates a new Photon image, based on the `src` image. The `src` image is duplicated: the function allocates a new palette, a new alpha map, etc.

After an image is allocated with `alloc_image()`, you are free to modify it. If you need to allocate memory for the image palette, image data, or alpha map, use the `alloc_mem()` function.

The newly allocated image will have the `flags` member set to Ph_RELEASE_IMAGE_ALL, meaning that all the fields of the new image that required allocating memory will be released when the image is released.

You should free the image with the PhAB exported `free_image()` function.

**Returns:**

A pointer to the image, or NULL when an error occurs.

**Classification:**

Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

**See also:**

`alloc_mem()`, `free_image()`, PhABResExportCommon_t, PhImage_t
PhABResExportCommon_t.alloc_mem()

A function exported by PhAB to allocate memory for the resource editor’s copy of some data

Synopsis:

```
char* alloc_mem ( int size )
```

Arguments:

```
size    The size, in bytes, of the buffer you want allocated.
```

Description:

This function is exported from PhAB in the PhABResExportCommon_t structure. This function can be used by a resource editor to allocate memory for its copy of some data. This memory should be freed with the PhAB exported free_mem() function.

Returns:

A pointer to the allocated memory, or NULL if an error occurred.

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

free_mem() PhABResExportCommon_t
A function exported by PhAB to answer a call to a `ResPluginAnyChangesF_t` function

**Synopsis:**

```c
void answer_changes ( 
    PhABHandle_t phab, 
    int answer, 
    int n, 
    void *pvalue )
```

**Arguments:**

- **phab** A handle, provided by PhAB as the `phab` argument when the plugin is created with `ResPluginFrugalCreateF_t` or `ResPluginFullCreateF_t`.
- **answer** Specifies whether the resource editor contains changes. Can be:
  - `RESPLUGIN_CHANGES` — plugin contains changes
  - `RESPLUGIN_NO_CHANGES` — plugin doesn’t contain changes
- **n** The new value. Its value depends on the datatype.
- **value** The new value. Its value depends on the datatype.

**Description:**

This function is exported from PhAB in the `PhABResExportFull_t` structure

This function is called from the full resource editor after the editor returns `RESPLUGIN_WAIT` to a call to the function defined by `ResPluginAnyChangesF_t` from PhAB. It should specify whether the editor contains any changes to its data. If there are changes, the plugin should use `n` and `value` to pass the new data back to PhAB.

**Classification:**

Photon

**Safety**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

- `PhABResExportFull_t.ResPluginAnyChangesF_t`
- `ResPluginFrugalCreateF_t.ResPluginFullCreateF_t`
PhABResExportCommon_t.apply()
Function exported by PhAB to apply changes in the resource editor

Synopsis:

```c
int apply (  
    PhABHandle_t phab,  
    int n,  
    void * value )
```

Arguments:

- `phab` A handle, provided by PhAB as the `phab` argument when the plugin is created with `ResPluginFrugalCreateF_t` or `ResPluginFullCreateF_t`.
- `n` Resource data, depending on the data type.
- `value` Resource data, depending on the data type.

Description:

This function is exported from PhAB in the `PhABResExportCommon_t` structure.

This function is called from the plugin when resource data changes have to be applied to the selected widget or widgets, for example, when a user clicks the `Apply` button in the resource editor.

The data value is represented by the parameter pair `n` and `value`. How these parameters are passed depends on the data type of the resource. See the description of resource data types for a discussion of how these parameters are passed.

The return value of this function indicates whether PhAB has taken control over the resource data. This prevents memory leaks when the master copy rule applies. When the `apply()` function returns zero, PhAB has taken the given value and has made it the master copy. When the `apply()` function returns non zero, PhAB has not made the given value the master copy, it has used the default value instead. In this case the plugin should update its internal structures by freeing any data it allocated for the value and using the default value instead.

Returns:

- `0` PhAB has made the resource data represented by `value` and `n` the master copy.
- `Non 0` PhAB has not made a master copy and is using the default value instead.

Classification:

- Photon
### Safety

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

PhABResExportCommon_t.ResPluginFrugalCreateF_t,
ResPluginFullCreateF_t.
PhABResExportFull_t.closing()

Function exported by PhAB to inform PhAB that the resource editor is closing

Synopsis:

```c
void closing ( PhABHandle_t phab )
```

Arguments:

- `phab` A handle, provided by PhAB as the `phab` argument when the plugin is created with `ResPluginFullCreateF_t`.

Description:

This function is exported from PhAB in the `PhABResExportFull_t` structure.

This function is called from the full resource editor to inform PhAB that the editor wants to initiate the closing procedure. This function should not be called from inside the resource editor’s destroy function, since in this case PhAB initiated the closing procedure.

Closing the plugin window may be triggered by a user clicking a `Cancel` button or the window’s close button. This function should be called before the closing occurs, because PhAB may cancel the closing action.

Classification:

- **Photon**

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

- `PhABResExportFull_t`, `ResPluginAnyChangesF_t`
- `ResPluginFullCreateF_t`.
PhABResExportCommon_t cmp()

A function exported by PhAB to compare two resource editor values

Synopsis:

```c
int cmp (  
    ResDatatype_t type,  
    int n1,  
    void *v1,  
    int n2,  
    void *v2 );
```

Arguments:

type  
The data type of the values you want to compare.

n1, v1  
The value members for the first value you want to compare. The values depend on the data type.

n2, v2  
The value members for the second value you want to compare. The values depend on the data type.

Description:

This function is exported from PhAB in the PhABResExportCommon_t structure.

This is a convenience function that you can use to compare two values. While in some cases, for instance with the RES_DATATYPE_NUMERIC datatype, it might be easy enough to just compare the values without using this function. In other cases, as with RES_DATATYPE_PIXMAP, it is easier to use this function.

Returns:

0  
the values are equal

non-zero  
the values are not equal

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PhABResExportCommon_t
Synopsis:

```c
PhabResWindowHandle_t create_window (    
    PhArea_t *area, 
    char *caption, 
    unsigned int flags, 
    PtWidget_t **container, 
    ResPluginActionNotifyF_t *notify, 
    void *notify_data )
```

Arguments:

- **area**: The area of the window, represented by a `PhArea_t`.
- **caption**: The window’s caption string, related to the resource being currently edited.
- **flags**: Additional information about how the window should look. Set to `RESPLUGIN_FIXED_SIZE` to make the resulting window non-resizeable.
- **container**: Filled by the `create_window()` function with a pointer to a container widget. You should use this widget pointer as the parent for whatever widgets are needed to render the full resource editor.
- **notify**: A pointer to a callback that notifies the resource editor of changes to the window. This callback has the following synopsis:

  ```c
typedef void ResPluginActionNotifyF_t (    
    ResPluginAction_t action, void *notify_data );
```

  Action is returned as one of:

  - `RESPLUGIN_ACTION_DEFAULT` — the resource editor user has requested the default action.
  - `RESPLUGIN_ACTION_CLOSE` — the resource editor user has requested to close the plugin’s window.
  - `RESPLUGIN_ACTION_APPLY` — the resource editor user has requested the apply action.

- **notify_data**: A pointer to data that you want to pass as the second argument to the `notify` function.

Description:

This function is exported from PhAB in the `PhABResExportFull_t` structure.

This convenience function should be called from within the full editor’s create function to create a typical resource editor window. The function creates the window with all the necessary callbacks already attached and will notify the plugin about various actions using the `notify` function and `notify_data`. 
Returns:

A `PhabResWindowHandle_t` which has to be used in the related windowing functions

- success
- NULL error

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

`destroy()`, `get_area()`, `get_res()`, `PhABResExportFull_t`, `set_state()`, `to_front()`.
PhABResExportFull_t.destroy()

A function exported by PhAB to destroy a window created by create_window()

Synopsis:

```c
void destroy (  
   PhabResWindowHandle_t wnd )
```

Arguments:

wnd: The window handle for the window you want to destroy, originally returned by PhABExportCreateWindowF_t create_window().

Description:

This function is exported from PhAB in the PhABResExportFull_t structure.

This convenience function should be called from the full editor to destroy the convenience window wnd, created by calling create_window(). The only time this function should be called is from the destroy method of the full resource editor.

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

See also:

create_window(), get_area(), get_res(), PhABResExportFull_t, set_state(), to_front().
PhABResExportCommon_t.dup()

A PhAB exported function to duplicate a value

Synopsis:

```c
void dup (  
    ResDatatype_t type,  
    int n,  
    void *v,  
    int *dest_n,  
    void **dest_value );
```

Arguments:

- `type` The data type of the data you want to duplicate.
- `n, v` The source value you want to duplicate. The values depend on the data type.
- `dest_n, dest_value` The destination for value you want to duplicate. The values depend on the data type.

Description:

This function is exported from PhAB in the PhABResExportCommon_t structure. This is a convenience function that you can use to duplicate a value.

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

PhABResExportCommon_t
PhABResExportCommon_t.free()

A function exported by PhAB to free a value allocated with

Synopsis:

```c
void free (  
    ResDatatype_t type,  
    int n,  
    void *v );
```

Arguments:

- `type` The type of the data you want to free.
- `n` The n value for the data you want to see. This value depends on the data type.
- `v` The v value for the data you want to see. This value depends on the data type.

Description:

Exported from PhAB in the PhABResExportCommon_t structure.

This is a convenience function that you can use to free a value.

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

See also:

`alloc_mem()`, PhABResExportCommon_t.
PhABResExportCommon_t.free_image() © 2007, QNX Software Systems GmbH & Co. KG.

A function exported by PhAB to free an image created with alloc_image()

Synopsis:

    void free_image ( PhImage_t *p )

Arguments:

    p  A pointer to the image date you want to free, returned by alloc_image().

Description:

This function is exported from PhAB in the PhABResExportCommon_t structure.
This function should be used by the plugin to free the image allocated using the PhAB exported alloc_image() function.
Because the flags member is set to Ph_RELEASE_IMAGE_ALL when an image is allocated with alloc_image(), all the fields of the new image that required allocating memory are released when the image is released.

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
</tr>
<tr>
<td>Signal handler</td>
</tr>
<tr>
<td>Thread</td>
</tr>
</tbody>
</table>

See also:

    alloc_image(), PhABResExportCommon_t
**PhABResExportCommon_t.free_mem()**

A function exported by PhAB to free memory allocated by alloc_mem()

**Synopsis:**

```c
void free_mem ( char *p )
```

**Arguments:**

- `p` The pointer to the memory buffer returned by `alloc_mem()`.

**Description:**

This function is exported from PhAB in the PhABResExportCommon_t structure.

This function should be used by the resource editor to free the memory allocated using the `alloc_mem()`.

**Classification:**

<table>
<thead>
<tr>
<th>Safety</th>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

**See also:**

`alloc_mem()`, PhABResExportCommon_t.
A function exported by PhAB to get the area of a window created by `create_window()`

Synopsis:

```c
void get_area (  
    PhabResWindowHandle_t wnd,  
    PhArea_t *area)
```

Arguments:

- `wnd` is the window handle returned by `PhABExportCreateWindowF_t create_window()`.  
- `area` is filled in by the function with the area of the resource editor window.

Description:

This function is exported from PhAB in the `PhABResExportFull_t` structure. You can call this function from the full editor to find out the current area of the window `wnd`.

Classification:

- **Photon**

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

`create_window()`, `destroy()`, `get_res()`, `PhABResExportFull_t`, `set_state()`, `to_front()`.
PhABResExportCommon_t.get_res()
A function exported by PhAB to get the resource structure for another datatype

Synopsis:

```c
void get_res (  
ResDatatype_t type  
char *abstract_name_editor  
ResPluginFrugalEditor_t **pfrugal_editor,  
ResPluginFullEditor_t **pfull_editor  
)
```

Arguments:

- **type**: The datatype for the resource editor that you want to obtain a handle for.
- **abstract_name_editor**: The name of the resource editor you want to obtain a handle for. Use NULL to get the handle for the pre-defined editor. Otherwise, use a name for an editor defined in the palette files.
- **pfrugal_editor**: This argument is filled in by the function with a pointer to a ResPluginFrugalEditor_t, which represents the frugal editor that matches the `type` and `abstract_name_editor`. Pass NULL in this parameter if you're not interested in obtaining this pointer.
- **pfull_editor**: This argument is filled in by the function with a pointer to a ResPluginFullEditor_t, which represents the full editor that matches the `type` and `abstract_name_editor`. Pass NULL in this parameter if you're not interested in obtaining this pointer.

Description:

This function is exported from PhAB in the PhABResExportCommon_t structure.

This convenience function can be called from either the frugal or full editor to obtain the handle for the editor of another resource type. This allows one plugin to call another currently loaded resource editor plugin as a sub-editor. For example, a pixmap full editor needs to be able to edit the color of a pixel. To do this, you can use `get_res()` in the pixmap editor to create an instance of the full color editor.

Examples:

Here is a partial example of calling the color editor from a different editor plugin.

```c
/* taken from the pixmap editor plugin */
/* here "instance" is an instance to the full resource pixmap editor */
/* get the predefined resource editor for "color" */

instance->exp->common.get_res( RES_DATATYPE_COLOR, NULL, NULL,  
&instance->def_res_full_color );

/* did we obtain a valid pointer to the full resource editor for "color" ? */
```
if( instance->def_res_full_color ) {
    /* use the exported convenience functions that were passed for the pixmap
     * plugin editor */
    instance->color_editor_exp = *instance->exp;

    /* defined our own apply and closing functions */
    instance->color_editor_exp.common.apply = color_res_apply;
    instance->color_editor_exp.closing = color_res_closing;
    instance->color_editor_handle = instance->def_res_full_color->create(
        (PhABHandle_t) instance, &instance->color_editor_exp, NULL, color, NULL,
        NULL, "the color editor", color, NULL );
}

static int color_res_apply( PhABHandle_t phab, int color, void *value ) {
    PluginFullInstance_t *instance = (PluginFullInstance_t *) phab;
    /* here "instance" is an instance to the full resource pixmap editor */
    .......
    return 1;
}

static int color_res_closing( PhABHandle_t phab ) {
    PluginFullInstance_t *instance = (PluginFullInstance_t *) phab;
    /* here "instance" is an instance to the full resource pixmap editor */
    instance->color_editor_handle = NULL;
    return 1;
}

Classification:

Photon

Safety

<table>
<thead>
<tr>
<th>Interrupt handler</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

create_window(), destroy(), get_area(), PhABResExportCommon_t, set_state(),
to_front().
PhABResExportCommon_t.message()
A function exported by PhAB to display a message dialog

Synopsis:

```c
int message (  
    char *message,  
    char *button1,  
    char *button2,  
    char *button3,  
    int default_button );
```

Arguments:

- `message`: The text that appears in the message dialog.
- `button1`: The text for the first button.
- `button2`: The text for the second button. Set to NULL if you do not require a second button.
- `button3`: The text for the third button. Set to NULL if you do not require a third button.
- `default_button`: The index (1, 2, or 3) of the button with focus when the message dialog is displayed.

Description:

Exported from PhAB in the PhABResExportCommon_t structure.

This function displays a customizable message dialog.

Returns:

The index of the button the user clicked.

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

- PhABResExportCommon_t
**Synopsis:**

```c
void open ( PhABHandle_t phab )
```

**Arguments:**

- **phab**: A handle, provided by PhAB as the `phab` argument when the plugin is created with `ResPluginFrugalCreateF_t`.

**Description:**

This function is exported from PhAB in the `PhABResExportFrugal_t` structure. Call this function from the frugal editor to display the full editor. For example, the frugal editor might be display only, not allowing the user to change the resource data. However, the frugal editor might contain a button or area which the user can click to open the full editor.

**Classification:**

- **Photon**

  **Safety**

  - Interrupt handler: No
  - Signal handler: No
  - Thread: No

**See also:**

`ResPluginFrugalCreateF_t`
PhABResExportCommon_t
Functions exported by PhAB called by a full or frugal editor

Synopsis:

typedef struct PhABResExportCommon_ {
    PhABExportApplyF_t *apply;
    PhABExportAllocMemF_t *alloc_mem;
    PhABExportAllocImageF_t *alloc_image;
    PhABExportFreeMemF_t *free_mem;
    PhABExportFreeImageF_t *free_image;
    PhABExportCmpF_t *cmp;
    PhABExportDupF_t *dup;
    PhABExportFreeF_t *free;
    PhABExportGetResF_t *get_res;
    PhABExportMessageF_t *message;
} PhABResExportCommon_t;

Description:

This structure is a superclass for PhABResExportFrugal_t and PhABResExportFull_t.
The structure has these members:

apply A pointer to the apply() function exported by PhAB.
alloc_mem A pointer to the alloc_mem() function exported by PhAB.
alloc_image A pointer to the alloc_image() function exported by PhAB.
free_mem A pointer to the free_mem() function exported by PhAB.
free_image A pointer to the free_image() function exported by PhAB.
cmp A pointer to the cmp() function exported by PhAB.
dup A pointer to the dup() function exported by PhAB.
free A pointer to the free() function exported by PhAB.
get_res A pointer to the get_res() function exported by PhAB.
message A pointer to the message() function exported by PhAB.

Classification:

Photon
See also:

alloc_image(), alloc_mem(), apply(), cmp(), dup(), free(), free_image(), free_mem(),
get_res(), message(), PhABResExportFrugal_t, PhABResExportFull_t.
PhABResExportFrugal_t
A structure that defines the functions exported by PhAB that can be called by a frugal editor

Synopsis:

typedef struct PhABResExportFrugal_ {
    PhABResExportCommon_t common;
    PhABExportOpenF_t *open;
} PhABResExportFrugal_t;

Description:
When a new frugal resource editor is created, it is passed a pointer to this structure, which gives it access to the functions exported by PhAB.

The structure has these members:

common A PhABResExportCommon_t structure that points to functions common to frugal and full editors.
open A pointer to the open() function exported by PhAB.

Classification:
Photon

See also:
open(), PhABResExportCommon_t.
PhABResExportFull_t

A structure that defines the functions exported by PhAB that can be called by a full editor

Synopsis:

```c
typedef struct PhABResExportFull_t {
    PhABResExportCommon_t common;
    PhABExportClosingF_t *closing;
    PhABExportAnswerChangesF_t *answer_changes;
    PhABExportCreateWindowF_t *create_window;
    PhABExportSetStateF_t *set_state;
    PhABExportToFrontF_t *to_front;
    PhABExportGetAreaF_t *get_area;
    PhABExportDestroyF_t *destroy;
} PhABResExportFull_t;
```

Description:

When a new full resource editor is created, it is passed a pointer to this structure, which gives it access to the functions exported by PhAB.

The structure has these members:

- **common**: A `PhABResExportCommon_t` structure that points to functions common to frugal and full editors.
- **closing**: A pointer to the `closing()` function exported by PhAB.
- **answer_changes**: A pointer to the `answer_changes()` function exported by PhAB.
- **create_window**: A pointer to the `create_window()` function exported by PhAB.
- **set_state**: A pointer to the `set_state()` function exported by PhAB.
- **to_front**: A pointer to the `to_front()` function exported by PhAB.
- **get_area**: A pointer to the `get_area()` function exported by PhAB.
- **destroy**: A pointer to the `destroy()` function exported by PhAB.

Classification:

Photon

See also:

- `answer_changes()`, `closing()`, `create_window()`, `destroy()`, `get_area()`,
- `PhABResExportCommon_t.set_state()`, `to_front()`.
ResPlugin_t

A structure that defines all the resource editor plugins in a DLL

Synopsis:

```c
typedef struct ResPlugin_ {
    ResPluginLoadingF_t *loading;
    ResPluginUnloadingF_t *unloading;
    int n_full_editors;
    const ResPluginFullEditor_t* full_editors;
    int n_frugal_editors;
    const ResPluginFrugalEditor_t* frugal_editors;
} ResPlugin_t;
```

Description:

This structure defines the resource editor plugins in a DLL. An instance of this structure is pointed to by the `tab` member of the `AOInterfaces_t interfaces` array.

The members are:

- **loading**: A pointer to a `ResPluginLoadingF_t` function for the DLL, which is executed after the DLL is loaded into PhAB’s memory space.
- **unloading**: A pointer to a `ResPluginUnloadingF_t` function for the DLL, which is executed just before the DLL is unloaded from PhAB’s memory space.
- **n_full_editors**: The number of entries in the `full_editors` array.
- **n_frugal_editors**: The number of entries in the `frugal_editors` array.
- **full_editors**: An array of `ResPluginFullEditor_t` structures. Each structure in the array defines the API for a full resource editor in the DLL.
- **frugal_editors**: An array of `ResPluginFrugalEditor_t` structures. Each structure in the array defines the API for a frugal resource editor in the DLL.

Classification:

Photon

See also:

- `ResPluginLoadingF_t`, `ResPluginFullEditor_t`
- `ResPluginUnloadingF_t`
- `ResPluginFrugalEditor_t`
ResPluginAnyChangesF_t

A function you must write to indicate there are changes in the resource editor data

Synopsis:

typedef int ResPluginAnyChangesF_t( ResPluginHandle_t handle )

Description:

This function is exported in the ResPluginFullEditor_t structure.

This function is called by PhAB to ask the full editor whether it contains changes or not. If there are no changes, the plugin should return RESPLUGIN_NO_CHANGES. If there are changes, the plugin should return RESPLUGIN_CHANGES.

The plugin can also return RESPLUGIN_WAIT if it cannot answer right away. In this case the plugin has to call back later into PhAB, using the answer_changes() function. PhAB will not destroy the plugin instance until the call to answer_changes() is made. The RESPLUGIN_WAIT can be used in the case where the plugin spawns an external application to edit the resource, as described in the Plugin lifecycle section.

Arguments:

handle The ResPluginHandle_t handle returned when the instance of the editor is created with ResPluginFullCreateF_t.

Should return:

This function should return one of:

- RESPLUGIN_NO_CHANGES — there are no changes.
- RESPLUGIN_CHANGES — there are changes
- RESPLUGIN_WAIT — the plugin doesn’t know if there are changes or not and it will answer later by calling answer_changes().

Examples:

This sample function is from the complete plugin example at the end of this chapter.

```c
static int plugin_full_any_changes( ResPluginHandle_t handle )
{
    PluginFullInstance_t *instance = ( PluginFullInstance_t * ) handle;
    char *p;
    PtGetResource( instance->full_widget, Pt_ARG_TEXT_STRING, &p, 0 );
    if( !strcmp( p, instance->value_master ) ) return RESPLUGIN_NO_CHANGES;
    return RESPLUGIN_CHANGES;
}
```

Classification:

QNX Neutrino
See also:

`answer_changes()`, `ResPluginFrugalCreateF_t`, `ResPluginFullCreateF_t`, `ResPluginFullEditor_t`. 
A function you must write to temporarily block input

Synopsis:

```c
typedef void ResPluginBlockF_t ( 
    ResPluginHandle_t handle, 
    int block )
```

Description:

This function is exported in the `ResPluginFullEditor_t` structure.

This function is called by PhAB to indicate that the plugin should temporary block input to the resource editor (when the `block` parameter is not 0) or that the plugin should restore the input to the full resource editor (when the `block` parameter is 0).

When PhAB requests that the resource editor be blocked, you should set the Pt_BLOCKED flag on the window that contains the editor. If the PhAB exported `create_window()` function was used to create the editor’s window, then the PhAB exported `set_state()` function should be used with an argument equal to RESPLUGIN_STATE_BLOCKED to block the editor’s window or with an argument equal to RES_PLUGIN_STATE_UNBLOCKED to unblock it.

Arguments:

- **handle**: The `ResPluginHandle_t` handle returned when the instance of the editor is created with `ResPluginFullCreateF_t`.
- **block**: Indicates whether the plugin should block or unblock input. It can be:
  - 0 — unblock input
  - not 0 — block input

Examples:

This sample block function for a full editor is from the complete plugin example at the end of this chapter.

```c
static void plugin_full_block( ResPluginHandle_t handle, int block )
{
    PluginFullInstance_t *instance = ( PluginFullInstance_t * ) handle;
    instance->exp->set_state( instance->convenience_handle,
        block ? RESPLUGIN_STATE_BLOCKED :
            RESPLUGIN_STATE_UNBLOCKED );
}
```

Classification:

QNX Neutrino

See also:

`create_window()`, `ResPluginFrugalCreateF_t`, `ResPluginFullEditor_t`, `ResPluginFullCreateF_t`, `set_state()`. 
ResPluginCommon_t
A superclass structure that defines the common elements of a frugal or full resource editor

Synopsis:

typedef struct ResPluginCommon_ {
    const char *name;
    ResDatatype_t datatype;
    int datatype_version;

    /* the common functions */
    const ResPluginSetDataF_t *set_data;
    const ResPluginDestroyF_t *destroy;
} ResPluginCommon_t;

Description:

This is a structure that contains elements common to a frugal editor and full editor, and
is the superclass for the structures ResPluginFrugalEditor_t and
ResPluginFullEditor_t.

The members are:

name Uniquely identifies the plugin in the frugal_editors or full_editors array. It has to be unique in the array.
datatype The type of resource the editor is designed to edit. ResDatatype_t is defined as:

typedef enum { RES_DATATYPE_STRING,
    RES_DATATYPE_MULTI, RES_DATATYPE_CODE,
    RES_DATATYPE_FONT, RES_DATATYPE_COLOR,
    RES_DATATYPE_DOUBLE, RES_DATATYPE_NUMERIC,
    RES_DATATYPE_CHOICE, RES_DATATYPE_FLAG,
    RES_DATATYPE_PATTERN, RES_DATATYPE_LIST,
    RES_DATATYPE_PIXMAP, RES_DATATYPE_ALLOC } ResDatatype_t;
datatype_version The version of the datatype that this plugin understands. All the
datatyps, as described in this document, have the version 1,
defined as the constant RESPLUGIN_DATATYPE_VERSION:

#define RESPLUGIN_DATATYPE_VERSION 1

Plugins should use this constant as the datatype_version. For instance, it means that a RES_DATATYPE_PIXMAP editor
understands the pixmap type version 1, as described in this
chapter. It’s possible that in the future there will be changes to
some data types. For example, if the definition of PgColor_t
changed, then RES_DATATYPE_COLOR would require a version
2 editor to handle the changes.

set_data; A pointer to the ResPluginSetDataF_t function for the
resource editor.
destroy; A pointer to the ResPluginDestroyF_t function for the
resource editor.
Classification:

Photon

See also:

ResPluginDestroyF_t, ResPluginFrugalEditor_t,
ResPluginFullEditor_t, ResPluginSetDataF_t.
ResPluginDestroyF_t
A function you must write to release resources allocated to the plugin

Synopsis:

typedef void ResPluginDestroyF_t (ResPluginHandle_t handle )

Description:

This function is exported in the ResPluginCommon_t structure.

This function is called by PhAB to destroy an instance of the resource editor and free
the resources allocated.

If the plugin is a frugal editor, then the plugin’s widgets have already been destroyed at
this point.

If the plugin is a full editor, then it is the plugin’s responsibility to destroy its widgets
and window, perhaps by calling PtDestroyWidget() with the topmost widget as an
argument.

Arguments:

handle  The ResPluginHandle_t handle returned when the instance of the editor
is created with a create function, either ResPluginFrugalCreateF_t or
ResPluginFullCreateF_t.

Examples:

This example destroy function is from the complete plugin example at the end of this
chapter.

static void plugin_full_destroy( ResPluginHandle_t handle )
{
    PluginFullInstance_t *instance =
        ( PluginFullInstance_t * ) handle;
    instance->exp->destroy( instance->convenience_handle );
    free( instance );
}

Classification:

QNX Neutrino

See also:

ResPluginCommon_t, ResPluginFrugalCreateF_t,
ResPluginFullCreateF_t, Resource datatypes.
A function you must write to disable the plugin

Synopsis:

```c
typedef void ResPluginDisableF_t ( ResPluginHandle_t handle )
```

Description:

This function is exported in the `ResPluginFullEditor_t` structure.

This function is called by PhAB to indicate that the resource editor should change its appearance so that it looks disabled, or greyed out. The editor should keep a Cancel button active even when the content and other buttons are disabled, so that a user can dismiss the window. The content should be changed back to the active state when PhAB sends new data to the editor via a call to the set data function. If the PhAB exported `create_window()` function was used to create the editor’s window, then the PhAB exported `set_state()` function should be used the the argument `RESPLUGIN_STATE_DISABLED`.

Arguments:

handle The `ResPluginHandle_t` handle returned when the instance of the editor is created with a create `ResPluginFullCreateF_t`.

Examples:

This sample disable function for a full editor is from the complete plugin example at the end of this chapter.

```c
static void plugin_full_disable( ResPluginHandle_t handle )
{
    PluginInstance_t *instance = ( PluginInstance_t * ) handle;
    PtArg_t args[3];
    instance->disabled = 1;
    PtSetArg( &args[0], Pt_ARG_FLAGS, Pt_GHOST|Pt_BLOCKED, Pt_GHOST|Pt_BLOCKED );
    PtSetArg( &args[1], Pt_ARG_CURSOR_TYPE, Ph_CURSOR_NOINPUT, 0 );
    PtSetArg( &args[2], Pt_ARG_CURSOR_OVERRIDE, 1, 0 );
    PtSetResources( instance->full, 3, args );
    PtSetResource( instance->full_widget, Pt_ARG_FLAGS, Pt_GHOST|Pt_BLOCKED, Pt_GHOST|Pt_BLOCKED );
    instance->exp->set_state( instance->convenience_handle, RESPLUGIN_STATE_DISABLED );
}
```

Classification:

QNX Neutrino

See also:

`ResPluginFullCreateF_t`, `ResPluginFullEditor_t`
Synopsis:

typedef ResPluginHandle_t
ResPluginFrugalCreateF_t (  
    PhABHandle_t phab ,  
    const PhABResExportFrugal_t *exp,  
    const ResPluginFormatData_t *format,  
    int n_default,  
    const void *default_value,  
    PtWidget_t *parent,  
    int n,  
    const void *value )

Description:

This function is exported in the ResPluginFrugalEditor_t structure.

This function is called to create a new instance of the frugal resource editor.

The plugin should create any necessary widgets inside the container parent widget.
The plugin must try to create its widget(s) with a transparent background, if possible,
and it should anchor its widgets or attach a resize callback to the parent in order
to properly display itself when the width of the parent changes. The parent widget will
have a non-transparent background.

If necessary, format information is passed by format when the plugin instance is
created. The format data depends on the data type. For example, for the
RES_DATATYPE_FLAG datatype, format provides the plugin with the list of possible
flag values, the flag manifests, and the associated masks, if any.

Arguments:

phab A PhABHandle_t provided by PhAB, and which should be used as
the phab argument when the plugin calls any of the functions
present in the exp table of functions.

exp A pointer to a PhABResExportFrugal_t defining the functions
exported by PhAB for the frugal editor. You can assume that the exp
pointer will remain valid during the lifetime of the plugin.

format A pointer to a ResPluginFormatData_t structure that describes
format information required by the data type. See Resource
datatypes for a description of how this structure applies to each
datatype.

n_default Default value for the resource, together with the default_value
argument.

default_value Default value for the resource. This pointer is guaranteed to remain
valid during the lifetime of the plugin.

parent The parent container widget for any widgets the resource editor
creates.
n  Initial resource data, depending on the data type. See Resource datatypes.

value  Initial resource data, depending on the data type. See Resource datatypes.

Should return:

This function should return a \texttt{ResPluginHandle\_t} plugin handle, or NULL on error. If you return a non-NULL value, this value will be passed as the first argument to all your other functions.

Examples:

This example create function for a frugal editor is from the complete plugin example at the end of this chapter.

```c
static ResPluginHandle\_t plugin\_frugal\_create
( PhABHandle\_t phab , const PhABResExportFrugal\_t *exp,
  const ResPluginFormatData\_t *format,
  int n\_default, const void *default\_value,
  PtWidget\_t *parent,
  int n, const void *value ) {

  PtArg\_t args[2];
  PhRect\_t offset = { { -1, -1 }, { -1, -1 } },
  PluginFrugalInstance\_t *instance;

  instance = calloc( 1, sizeof( *instance ) );
  if( !instance ) return NULL;

  instance->n\_default = n\_default;
  instance->default\_value = default\_value;
  instance->n\_master = n;
  instance->value\_master = value;

  instance->phab = phab;
  instance->exp = exp;

  PtSetArg( args[0], Pt\_ARG\_ANCHOR\_OFFSETS, &offset, 0 );
  PtSetArg( args[1], Pt\_ARG\_TEXT\_STRING, value, 0 );
  instance->frugal = ApCreateWidget( db, "string\_frugal", 0, 0, 2, args );
  PtAddCallback( instance->frugal, Pt\_CB\_TEXT\_CHANGED, plugin\_frugal\_changed, instance );

  return ( ResPluginHandle\_t ) instance;
}
```

Classification:

QNX Neutrino

See also:

\texttt{PhABResExportFrugal\_t, ResPluginFrugalEditor\_t, Resource datatypes.}
ResPluginFrugalEditor_t
A structure that defines a frugal editor

Synopsis:

typedef struct ResPluginFrugalEditor_ {
        /* the common stuff */
        ResPluginCommon_t common;

        /* the frugal editor */
        ResPluginFrugalCreateF_t *create;
} ResPluginFrugalEditor_t;

Description:

This structure defines the elements and functions for a frugal editor required for the frugal_editors array of ResPlugin_t.

ResPluginFrugalEditor_t has these members:

common A ResPluginCommon_t structure that defines name, datatype, version, and endian string (if required) for a frugal editor, as well as its set data and destroy functions.

create A pointer to the plugin’s ResPluginFrugalCreateF_t create() function.

Classification:

Photon

See also:

ResPlugin_t, ResPluginCommon_t.
ResPluginFullCreateF_t

A function you must write to create an instance of a full resource editor

Synopsis:

typedef ResPluginHandle_t
ResPluginFullCreateF_t( PhABHandle_t phab ,
    const PhABResExportFull_t *exp,
    const ResPluginFormatData_t *format,
    int n_default,
    const void *default_value,
    PhArea_t *area,
    char *caption,
    int n,
    const void *value )

Description:

This function is exported in the ResPluginFullEditor_t structure.

This function is called to create a new instance of the full resource editor.

If necessary, format information is passed by format when the plugin instance is created. The format data depends on the data type. For example, for the RES_DATATYPE_FLAG datatype, format provides the plugin with the list of possible flag values, the flag manifests, and the associated masks, if any.

The data value is represented by the parameter pair n and value. How these parameters are passed depends on the data type of the resource. See the description of resource data types for a discussion of how these parameters are passed.

To ensure a consistent look for the full resource editors, PhAB provides a library of functions to support the resource editor’s windowing requirements. For example, create_window() creates a typical resource editor window, complete with standard Cancel, Default, Apply, and Done buttons. The resource editor needs only to populate a container inside the typical window for its specific editing requirements.

You can choose to use these supporting functions or you can create your own window for the resource editor. You can even spawn an external application, pass it the data, and manage the external application as a resource editor in PhAB. This approach is described in the Plugin lifecycle section.

Arguments:

phab A PhABHandle_t provided by PhAB, and which should be used as the phab argument when the plugin calls any of the functions present in the exp table of functions.

exp A pointer to a PhABResExportFull_t defining the functions exported by PhAB for the full editor. You can assume that the exp pointer will remain valid during the lifetime of the plugin.

format A pointer to a ResPluginFormatData_t structure that describes format information required by the data type.

n_default Default value for the resource, together with the default_value argument.
default_value

Default value for the resource, together with \texttt{n\_default}. This pointer is guaranteed to remain valid during the lifetime of the plugin.

area

A pointer to a \texttt{PhArea\_t} representing the full resource editor area. PhAB may pass the area argument as NULL, meaning that it has no records about this resource editor, and in this case you should use the area that best fits the editor.

caption

A string related to the resource being currently edited. This string can be displayed as the window’s title or maybe in a label inside the window.

n

Initial resource data, depending on the data type.

value

Initial resource data, depending on the data type.

Should return:

This function should return a \texttt{ResPluginHandle\_t} plugin handle, or NULL on error. If you return a non-NULL value, this value is passed as the first argument to all your other functions.

Examples:

This sample create function for a full editor is from the complete plugin example at the end of this chapter.

```c
static ResPluginHandle_t plugin_full_create
    ( PhABHandle_t phab , const PhABResExportFull_t *exp,
      const ResPluginFormatData_t *format,
      int n_default, const void *default_value,
      PhArea_t *area, char *caption,
      int n, const void *value )
{
    PluginFullInstance_t *instance;
    PtWidget_t *parent;
    PhRect_t offset = { { 0, 0 }, { 0, 0 } };
    PhArea_t tarea;
    PtArg_t args[1];
    int start = 0, end = n;

    instance = calloc( 1, sizeof( *instance ) );
    if ( !instance ) return NULL;

    instance->n_default = n_default;
    instance->default_value = default_value;
    instance->n_master = n;
    instance->value_master = value;

    instance->phab = phab;
    instance->exp = exp;

    if ( !area ) {
      PhRect_t rect;
      PhWindowQueryVisible( 0, 0, 0, &rect );
      tarea.pos.x = rect.ul.x + 100;
      tarea.pos.y = rect.ul.y + 100;
      tarea.size.w = 340;
      tarea.size.h = 150;
      area = &tarea;
    }
    instance->convenience_handle = exp->create_window( phab, area, caption, 0, &parent);
```
```
plugin_full_notify, instance );

PtSetParentWidget( parent );
PSetArg( &args[0], Pt_ARG_ANCHOR_OFFSETS, &offset, 0 );
instance->full = ApCreateWidgetFamily( db, "string_full_container", 0, 0, 1, args );
instance->full_widget = PtWidgetChildFront( instance->full );
PtAddCallback( instance->full, Pt_CB_RESIZE, plugin_full_resize, instance );
PtAddCallback( instance->full_widget, Pt_CB_TEXT_CHANGED, plugin_full_changed, instance );
PtAddCallback( instance->full_widget, Pt_CB_ACTIVATE, plugin_full_done, instance );
PSetResource( instance->full_widget, Pt_ARG_TEXT_STRING, value, 0 );
PtTextSetSelection( instance->full_widget, &start, &end );
PtRealizeWidget( instance->full );
set_state( instance, value );
return ( ResPluginHandle_t ) instance;
```

**Classification:**

QNX Neutrino

**See also:**

`create_window()`, `PhABResExportFull_t`, `PhArea_t`, `ResPluginFullEditor_t`. 
### Synopsis:

```c
typedef struct ResPluginFullEditor_t {
    /* the common stuff */
    ResPluginCommon_t common;

    /* the full editor */
    ResPluginFullCreateF_t *create;
    ResPluginDisableF_t *disable;
    ResPluginBlockF_t *block;
    ResPluginToFrontF_t *to_front;
    ResPluginAnyChangesF_t *any_changes;
    ResPluginGetChangesF_t *get_changes;
    ResPluginGetAreaF_t *get_area;
} ResPluginFullEditor_t;
```

### Description:

This structure defines the elements and functions for a full editor required for the `full_editors` array of `ResPlugin_t`.

`ResPluginFullEditor_t` has these members:

- **common**: An instance of the `ResPluginCommon_t` structure that defines the common functions for this plugin.
- **create**: A pointer to the plugin’s `ResPluginFullCreateF_t` create function.
- **disable**: A pointer to the plugin’s `ResPluginDisableF_t` disable function.
- **block**: A pointer to the plugin’s `ResPluginBlockF_t` block function.
- **to_front**: A pointer to the plugin’s `ResPluginToFrontF_t` to front function.
- **any_changes**: A pointer to the plugin’s `ResPluginAnyChangesF_t` any changes function.
- **get_changes**: A pointer to the plugin’s `ResPluginGetChangesF_t` get changes function.
- **get_area**: A pointer to the plugin’s `ResPluginGetAreaF_t` get area function.

### Classification:

Photon
See also:

ResPluginAnyChangesF_t, ResPluginBlockF_t,
ResPluginCommon_t, ResPluginDisableF_t, ResPluginFullCreateF_t,
ResPluginGetAreaF_t, ResPluginGetChangesF_t, ResPluginToFrontF_t.
ResPluginGetAreaF_t

A function you must write to retrieve the area occupied by the full resource editor window

Synopsis:

```c
typedef void ResPluginGetAreaF_t (
    ResPluginHandle_t handle,
    PhArea_t *area )
```

Description:

This function is exported in the `ResPluginFullEditor_t` structure.

This function is called by PhAB to find out the current area of the full resource editor. The plugin should fill in the area pointer with the editor’s current area.

Arguments:

- `handle` - The `ResPluginHandle_t` handle returned when the instance of the editor is created with `ResPluginFullCreateF_t`.
- `area` - A pointer to a `PhArea_t` structure you fill in to define the current area of the resource editor.

Examples:

This sample `ResPluginGetAreaF_t` function for a full editor is from the complete plugin example at the end of this chapter.

```c
static void plugin_full_get_area( ResPluginHandle_t handle, PhArea_t *area )
{
    PluginFullInstance_t *instance = ( PluginFullInstance_t * ) handle;
    instance->exp->get_area( instance->convenience_handle, area );
}
```

Classification:

QNX Neutrino

See also:

`ResPluginFullCreateF_t, ResPluginFullEditor_t, PhArea_t`. 
ResPluginGetChangesF_t

A function you must write to get changed resource data

Synopsis:

typedef void ResPluginGetChangesF_t (  
    ResPluginHandle_t handle,  
    int *pn,  
    void **pvalue )

Description:

This function is exported in the ResPluginFullEditor_t structure.

This function is called by PhAB to retrieve changes to resource data from the editor. It is called only after a call to ResPluginAnyChangesF_t indicates that there are changes to the resource editor’s data.

If the master copy rule applies, PhAB will take this data and make it the new master copy.

Arguments:

handle The ResPluginHandle_t handle returned when the instance of the editor is created with ResPluginFullCreateF_t.

pn A pointer to resource data. The actual data depends on the resource type.

pvalue A pointer to resource data. The actual data depends on the resource type.

Examples:

This sample ResPluginGetChangesF_t function for a full editor is from the complete plugin example at the end of this chapter.

static void plugin_full_get_changes( ResPluginHandle_t handle, int *pn, void **pvalue )
{
    PluginFullInstance_t *instance = ( PluginFullInstance_t * ) handle;
    char *p;
    PtGetResource( instance->full_widget, Pt_ARG_TEXT_STRING, &p, 0 );
    *pn = strlen( p );
    *pvalue = strdup( p );
}

Classification:

QNX Neutrino

See also:

ResPluginAnyChangesF_t, ResPluginFullCreateF_t, ResPluginFullEditor_t.
**ResPluginLoadingF_t**

A function you must write to initialize resources shared between plugins in a DLL

### Synopsis:

```c
typedef int ResPluginLoadingF_t (  
    const char *path  )
```

### Description:

This function is exported in the `ResPlugin_t` structure.

This function is called right after the DLL is loaded into PhAB’s address space, and should set up any resources that are common to the plugins found in the DLL. In particular, if you use PhAB to build a database of widgets that are going to be used to create instances of various plugins, you should initialize its `AbContext` structure by calling `ApAddContext( &AbContext, path)`.

Since it is possible to have different pathnames that point to the same file, this function may be called more than once, but it’s guaranteed that the corresponding `ResPluginUnloadingF_t` unloading function will be called the same number of times.

### Arguments:

- **path**  
  The path to the DLL that contains the plugin.

### Should return:

This function should return a non-zero value to indicate a serious failure. This will prevent PhAB from attempting to use the resource editors in the DLL.

### Examples:

This example loading function is from the complete plugin example at the end of this chapter.

```c
static int loaded_count = 0;
static ApDBase_t *db = NULL;

static int plugin_loading( const char *path ) {  
    if( loaded_count == 0 && ApAddContext( &AbContext, path ) )  
        return -1;

    db = ApOpenDBase( ABM_string_db );
    ++loaded_count;
    return 0;
}
```

### Classification:

QNX Neutrino

### See also:

ResPluginSetDataF_t

A function you must write to set the resource data in the editor

Synopsis:

```c
typedef void ResPluginSetDataF_t(
    ResPluginHandle_t handle,
    int n,
    const void *value,
    const ResPluginFormatData_t *format )
```

Description:

This function is exported in the `ResPluginCommon_t` structure.

This function is called by PhAB to set a data value in the plugin editor, after the plugin instance is created.

The data value is represented by the parameter pair `n` and `value`. How these parameters are passed depends on the data type of the resource. See the description of resource data types for a discussion of how these parameters are passed.

For some datatypes, PhAB will guarantee that it is safe for the plugin to look at the content pointed to by `value`, not only for the duration of this function, but beyond that. That pointer is called PhAB’s “master copy” and it is guaranteed to stay a valid pointer until the next call to the function.

If necessary, format information is passed by `format` when the plugin instance is created. The format data depends on the data type. For example, for the `RES_DATATYPE_FLAG` datatype, `format` provides the plugin with the list of possible flag values, the flag manifests, and the associated masks, if any.

In some circumstances, the format data might change for the same plugin instance. In this case PhAB calls the set data function again with a new `format` structure.

Arguments:

- `handle` The `ResPluginHandle_t` handle returned when the instance of the editor is created with a create function, either `ResPluginFrugalCreateF_t` or `ResPluginFullCreateF_t`.
- `n` Resource data, depending on the data type. See Resource datatypes.
- `value` Resource data, depending on the data type. See Resource datatypes.
- `format` A pointer to a `ResPluginFormatData_t` structure that describes format information required by the data type. See Resource datatypes for a description of how this structure applies to each datatype.

Examples:

This sample function is from the complete plugin example at the end of this chapter.

```c
static void plugin_frugal_set_data( ResPluginHandle_t handle, int n, const void *value,
    const ResPluginFormatData_t *format )
```

```c
    PluginFrugalInstance_t *instance =
        ( PluginFrugalInstance_t * ) handle;
```
ResPluginSetDataF_t

instance->n_master = n;
instance->value_master = value;
PtSetResource( instance->frugal, Pt_ARG_TEXT_STRING, value, 0 );
}

Classification:
QNX Neutrino

See also:
ResPluginCommon_t, ResPluginFrugalCreateF_t,
ResPluginFullCreateF_t, Resource datatypes.
ResPluginToFrontF_t

A function you must write to bring the resource editor window to front

Synopsis:

typedef void ResPluginToFrontF_t ( ResPluginHandle_t handle )

Description:

This function is exported in the ResPluginFullEditor_t structure.

This function is called by PhAB to indicate that the plugin should bring the full editor’s window to the front.

Arguments:

handle The ResPluginHandle_t handle returned when the instance of the editor is created with ResPluginFullCreateF_t.

Examples:

This sample function is from the complete plugin example at the end of this chapter.

static void plugin_full_to_front( ResPluginHandle_t handle )
{
    PluginFullInstance_t *instance = ( PluginFullInstance_t * ) handle;
    instance->exp->to_front( instance->convenience_handle );
}

Classification:

QNX Neutrino

See also:

ResPluginFrugalCreateF_t, ResPluginFullCreateF_t, ResPluginFullEditor_t.
ResPluginUnloadingF_t
A function you must write to release resources shared between plugins in a DLL

Synopsis:

```c
typedef void ResPluginUnloadingF_t( void )
```

Description:

This function is exported in the `ResPlugin_t` structure.

This function is called before the DLL is unloaded from PhAB’s address space, and should release resources set up in the `ResPluginLoadingF_t` loading function. Since the loading function may have been called more than once, there is a corresponding call to the unloading function for each successful call to the loading function.

Examples:

This example unloading function is from the complete plugin example at the end of this chapter.

```c
static int loaded_count = 0;
static ApDBase_t *db = NULL;

static void plugin_unloading( void ) {
  if( -- loaded_count == 0 ) {
    ApCloseDBase( db );
    db = NULL;
    ApRemoveContext( &AbContext );
  }
}
```

Classification:

QNX Neutrino

See also:

`ResPlugin_t, ResPluginLoadingF_t`
PhABResExportFull_t.set_state()

A function exported by PhAB to set the state of a window created by create_window()

Synopsis:

```c
int set_state (  
    PhabResWindowHandle_t wnd,  
    ResFullWindowState_t state );
```

Arguments:

- **wnd**: The window handle returned by PhABExportCreateWindowF_t create_window().
- **state**: The render action to take. Can be one of:
  - `RESPLUGIN_STATE_DISABLED` — the plugin has no data applied.
  - `RESPLUGIN_STATE_MATCH_DEFAULT` — the plugin data matches the default value for that resource.
  - `RESPLUGIN_STATE_MATCH_MASTER` — the plugin data matches PhAB’s value for that resource.
  - `RESPLUGIN_STATE_MATCH_DEFAULT_MASTER` — indicates that the plugin data matches both the default value and PhAB’s value for that resource.
  - `RESPLUGIN_STATE_NO_MATCH` — indicates that the data inside the plugin doesn’t match either the default value or PhAB’s value for that resource.
  - `RESPLUGIN_STATE_BLOCKED` — indicates that the plugin is blocked (has no input).
  - `RESPLUGIN_STATE_UNBLOCKED` — indicates that the plugin is unblocked (has input).

Description:

This function is exported from PhAB in the PhABResExportFull_t structure.

You can call this function from a full resource editor to render the window created in create_window() in a certain state.

Classification:

Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>
See also:

create_window(), destroy(), get_area(), get_res(), PhABResExportFull_t, to_front().
A function exported by PhAB to bring to the front a window created by create_window()

Synopsis:

```c
void to_front ( PhabResWindowHandle_t wnd )
```

Arguments:

- `wnd` The window handle returned by `PhABExportCreateWindowF_t create_window()`.

Description:

This function is exported from PhAB in the `PhABResExportFull_t` structure. Call this function from the full resource editor to bring its window to the front.

Classification:

- Photon

<table>
<thead>
<tr>
<th>Safety</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt handler</td>
<td>No</td>
</tr>
<tr>
<td>Signal handler</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>No</td>
</tr>
</tbody>
</table>

See also:

`create_window()`, `destroy()`, `get_area()`, `get_res()`, `PhABResExportFull_t`, `set_state()`.
Chapter 11

Resource editor plugin examples
This chapter contains examples of using the PhAB plugin API to create resource editors. There are two examples:

- Sample string editor — an example frugal and full editor for a string resource type
- Sample external editor — an example of using ped as an external resource editor

**A string editor**

The following is an example resource editor plugin for editing a string resource. This is taken from the actual PhAB source code. Not shown is the PhAB project which contains this source file and implements the widgets within a picture module.

```c
#include <Pt.h>
#include <Ap.h>

/* Local headers */
#include <res_plugin_api.h>
#include <sri/sri.h>
#include "abimport.h"
#include "proto.h"

typedef struct {
    int n_master;
    void *value_master;
    int n_default;
    void *default_value;
    PhABHandle_t phab;
    PhABResExportFrugal_t *exp;
    PtWidget_t *frugal;
} PluginFrugalInstance_t;

typedef struct {
    int n_master;
    void *value_master;
    int n_default;
    void *default_value;
    PhABHandle_t phab;
    PhABResExportFull_t *exp;
    PhabResWindowHandle_t convenience_handle;
    int disabled;  // initially zero
    PtWidget_t *full, *full_widget;
} PluginFullInstance_t;

static int plugin_loading(const char *path);
static void plugin_unloading(void);
static void plugin_frugal_set_data(ResPluginHandle_t handle, int n, const void *value,
                                 const ResPluginFormatData_t *format);
static void plugin_full_set_data(ResPluginHandle_t handle, int n, const void *value,
                                 const ResPluginFormatData_t *format);
static void plugin_frugal_destroy(ResPluginHandle_t handle);
static void plugin_full_destroy(ResPluginHandle_t handle);
static ResPluginHandle_t plugin_frugal_create(PhABHandle_t phab, const PhABResExportFrugal_t *exp,
                                              const ResPluginFormatData_t *format,
                                              int n_default, const void *default_value,
                                              PtWidget_t *parent, int n, const void *value);
static ResPluginHandle_t plugin_full_create(PhABHandle_t phab, const PhABResExportFull_t *exp,
                                             const ResPluginFormatData_t *format,
                                             int n_default, const void *default_value,
                                             PhArea_t *area, char *caption,
                                             int n, const void *value);
static void plugin_full_disable(ResPluginHandle_t handle);
static void plugin_full_block(ResPluginHandle_t handle, int block);
static void plugin_full_to_front(ResPluginHandle_t handle);
static int plugin_full_any_changes(ResPluginHandle_t handle);
static void plugin_full_get_changes(ResPluginHandle_t handle, int *pn,
```

September 10, 2007 Chapter 11 • Resource editor plugin examples 303
void **pvalue);

static void plugin_full_get_area( ResPluginHandle_t handle, PhArea_t *area );
static void plugin_full_notify( ResPluginAction_t action, void *data );

/* callbacks */
static int plugin_frugal_changed( PtWidget_t *widget, void *data, 
   PtCallbackInfo_t *cbinfo);
static int plugin_full_resize( PtWidget_t *widget, void *data, 
   PtCallbackInfo_t *cbinfo);
static int plugin_full_changed( PtWidget_t *widget, void *data, 
   PtCallbackInfo_t *cbinfo);
static int plugin_full_done( PtWidget_t *widget, void *data, 
   PtCallbackInfo_t *cbinfo);
static void set_state( PluginFullInstance_t *instance, char *value );

static ResPluginFullEditor_t full_editors[1] =
{
  {
    "string",
    RES_DATATYPE_STRING,
    RESPLUGIN_DATATYPE_VERSION,
    plugin_full_set_data,
    plugin_full_destroy
  },
  plugin_full_create,
  plugin_full_disable,
  plugin_full_block,
  plugin_full_to_front,
  plugin_full_any_changes,
  plugin_full_get_changes,
  plugin_full_get_area
};

static ResPluginFrugalEditor_t frugal_editors[1] =
{
  {
    "string",
    RES_DATATYPE_STRING,
    RESPLUGIN_DATATYPE_VERSION,
    plugin_frugal_set_data,
    plugin_frugal_destroy
  },
  plugin_frugal_create,
};

static ResPlugin_t tab =
{ 
  plugin_loading,
  plugin_unloading,
  1,
  full_editors,
  1,
  frugal_editors
};

A0Interface_t string_interfaces[] = {
  { "PhAB Resource Editors", RESPLUGIN_VERSION, &tab },
  { 0, 0, 0 }
};

static int loaded_count = 0;
static ApDBase_t *db = NULL;

static int plugin_loading( const char *path ) {
  if( loaded_count == 0 && ApAddContext( &AbContext, path ) )
    return -1;

  db = ApOpenDBase( ABM_string_db );
  ++loaded_count;
  return 0;
}
static void plugin_unloading( void ) {
    if ( --loaded_count == 0) {
        ApCloseDBase( db );
        db = NULL;
        ApRemoveContext( &AbContext );
    }
}

static ResPluginHandle_t plugin_frugal_create( PhABHandle_t phab , const PhABResExportFrugal_t *exp,
        const ResPluginFormatData_t *format,
        int n_default, const void *default_value,
        PtWidget_t *parent,
        int n, const void *value ) {

    PtArg_t args[2];
    PhRect_t offset = { { -1, -1 }, { -1, -1 } };
    PluginFrugalInstance_t *instance;

    instance = calloc( 1, sizeof( *instance ) );
    if ( !instance ) return NULL;

    instance->n_default = n_default;
    instance->default_value = default_value;
    instance->n_master = n;
    instance->value_master = value;

    instance->phab = phab;
    instance->exp = exp;

    PtSetArg( &args[0], Pt_ARG_ANCHOR_OFFSETS, &offset, 0 );
    PtSetArg( &args[1], Pt_ARG_TEXT_STRING, value, 0 );
    instance->frugal = ApCreateWidget( db, "string_frugal", 0, 0, 2, args );
    PtAddCallback( instance->frugal, Pt_CB_TEXT_CHANGED, plugin_frugal_changed, instance );
    return ( ResPluginHandle_t ) instance;
}

static void plugin_frugal_set_data( ResPluginHandle_t handle, int n, const void *value,
        const ResPluginFormatData_t *format ) {

    PluginFrugalInstance_t *instance = ( PluginFrugalInstance_t * ) handle;
    instance->n_master = n;
    instance->value_master = value;

    PtSetResource( instance->frugal, Pt_ARG_TEXT_STRING, value, 0 );
}

static int plugin_frugal_changed( PtWidget_t *widget, void *data,
        PtCallbackInfo_t *cbinfo ) {

    PluginFrugalInstance_t *instance = ( PluginFrugalInstance_t * ) data;
    char *p;
    PtGetResource( widget, Pt_ARG_TEXT_STRING, &p, 0 );
    instance->n_master = strlen( p );
    instance->value_master = strdup( p );
    if ( instance->exp->common.apply( instance->phab, instance->n_master,
            instance->value_master ) ) {
        /* we matched the default */
        free( instance->value_master );
        instance->value_master = instance->default_value;
        instance->n_master = instance->n_default;
    }
    return Pt_CONTINUE;
}

static void plugin_frugal_destroy( ResPluginHandle_t handle ) {

    free( ( PluginFrugalInstance_t * ) handle );
}

static ResPluginHandle_t plugin_full_create( PhABHandle_t phab , const PhABResExportFull_t *exp,
        const ResPluginFormatData_t *format,
        int n_default, const void *default_value,
PhArea_t *area, char *caption,
int n, const void *value )
{
  PluginFullInstance_t *instance;
  PtWidget_t *parent;
  PhRect_t offset = {{ 0, 0 }, { 0, 0 }};
  PhArea_t tarea;
  PtArg_t args[1];
  int start = 0, end = n;

  instance = calloc( 1, sizeof( *instance ) );
  if( !instance ) return NULL;

  instance->n_default = n_default;
  instance->default_value = default_value;
  instance->n_master = n;
  instance->value_master = value;

  instance->phab = phab;
  instance->exp = exp;

  if( !area ) {
    PhRect_t rect;
    PhWindowQueryVisible( 0, 0, 0, &rect );
    tarea.pos.x = rect.ul.x + 100;
    tarea.pos.y = rect.ul.y + 100;
    tarea.size.w = 340;
    tarea.size.h = 150;
    area = &tarea;
  }

  instance->convenience_handle = exp->create_window( phab, area, caption, 0,
                                                                                  parent, plugin_full_notify,
                                                                                  instance );

  PtSetParentWidget( parent );
  PtSetArg( &args[0], Pt_ARG_ANCHOR_OFFSETS, &offset, 0 );
  instance->full= ApCreateWidgetFamily( db, "string_full_container", 0, 0, 1, args );
  instance->full_widget = PtWidgetChildFront( instance->full );
  PtAddCallback( instance->full, Pt_CB_RESIZE, plugin_full_resize, instance );
  PtAddCallback( instance->full_widget, Pt_CB_TEXT_CHANGED, plugin_full_changed,
                                                                 instance );
  PtAddCallback( instance->full_widget, Pt_CB_ACTIVATE, plugin_full_done, instance );
  PtSetResource( instance->full_widget, Pt_ARG_TEXT_STRING, value, 0 );
  PtSetTextSelection( instance->full_widget, &start, &end );
  PtRealizeWidget( instance->full );
  set_state( instance, value);
  return ( ResPluginHandle_t ) instance; 
}

PtSetParentWidget( parent );
PtSetArg( args[0], Pt_ARG_ANCHOR_OFFSETS, &offset, 0 );
instance->full= ApCreateWidgetFamily( db, "string_full_container", 0, 0, 1, args );
instance->full_widget = PtWidgetChildFront( instance->full );
PtAddCallback( instance->full, Pt_CB_RESIZE, plugin_full_resize, instance );
PtAddCallback( instance->full_widget, Pt_CB_TEXT_CHANGED, plugin_full_changed,
                                                                  instance );
PtAddCallback( instance->full_widget, Pt_CB_ACTIVATE, plugin_full_done, instance );
PtSetResource( instance->full_widget, Pt_ARG_TEXT_STRING, value, 0 );
PtSetTextSelection( instance->full_widget, &start, &end );
PtRealizeWidget( instance->full );
set_state( instance, value);
return ( ResPluginHandle_t ) instance; 
}

static void plugin_full_destroy( ResPluginHandle_t handle )
{
  PluginFullInstance_t *instance = ( PluginFullInstance_t * ) handle;
  instance->exp->destroy( instance->convenience_handle );
  free( instance );
}

static void plugin_full_disable( ResPluginHandle_t handle )
{
  PluginFullInstance_t *instance = ( PluginFullInstance_t * ) handle;
  PtSetArg( args[0], Pt_ARG_FLAGS, Pt_GHOST| Pt_BLOCKED, Pt_GHOST| Pt_BLOCKED );
  PtSetArg( args[1], Pt_ARG_CURSOR_TYPE, Ph_CURSOR_NOINPUT, 0 );
  PtSetArg( args[2], Pt_ARG_CURSOR_OVERRIDE, 1, 0 );
  PtSetResource( instance->full, 3, args );
  PtSetResource( instance->full_widget, Pt_ARG_FLAGS, Pt_GHOST|Pt_BLOCKED, Pt_GHOST|Pt_BLOCKED );
  instance->exp->set_state( instance->convenience_handle, RESPLUGIN_STATE_DISABLED );
}

static void plugin_full_block( ResPluginHandle_t handle, int block )
{
PluginFullInstance_t *instance = ( PluginFullInstance_t * ) handle;
    instance->exp->set_state( instance->convenience_handle, block ?
        RESPLUGIN_STATE_BLOCKED : RESPLUGIN_STATE_UNBLOCKED );
}

static int plugin_full_any_changes( ResPluginHandle_t handle )
{
    PluginFullInstance_t *instance = ( PluginFullInstance_t * ) handle;
    char *p;
    PtGetResource( instance->full_widget, Pt_ARG_TEXT_STRING, &p, 0 );
    if( !strcmp( p, instance->value_master ) ) return RESPLUGIN_NO_CHANGES;
    return RESPLUGIN_CHANGES;
}

static void plugin_full_get_changes( ResPluginHandle_t handle, int *pn, void **pvalue )
{
    PluginFullInstance_t *instance = ( PluginFullInstance_t * ) handle;
    char *p;
    PtGetResource( instance->full_widget, Pt_ARG_TEXT_STRING, &p, 0 );
    *pn = strlen( p );
    *pvalue = strdup( p );
}

static void plugin_full_get_area( ResPluginHandle_t handle, PhArea_t *area )
{
    PluginFullInstance_t *instance = ( PluginFullInstance_t * ) handle;
    instance->exp->get_area( instance->convenience_handle, area );
}

static void plugin_full_to_front( ResPluginHandle_t handle )
{
    PluginFullInstance_t *instance = ( PluginFullInstance_t * ) handle;
    instance->exp->to_front( instance->convenience_handle );
}

static void plugin_full_set_data( ResPluginHandle_t handle, int n, const void *value,
    const ResPluginFormatData_t *format )
{
    PluginFullInstance_t *instance = ( PluginFullInstance_t * ) handle;
    int start = 0, end = n;
    PtSetResource( instance->full_widget, Pt_ARG_TEXT_STRING, value, 0 );
    PtTextSetSelection( instance->full_widget, &start, &end );

    instance->n_master = n;
    instance->value_master = value;
    set_state( instance, value );
    if( instance->disabled )
    {
        PtArg_t args[3];
        instance->disabled = 0;
        PtSetArg( &args[0], Pt_ARG_FLAGS, 0, Pt_GHOST|Pt_BLOCKED );
        PtSetArg( &args[1], Pt_ARG_CURSOR_TYPE, Ph_CURSOR_INHERIT, 0 );
        PtSetArg( &args[2], Pt_ARG_CURSOR_OVERRIDE, 0, 0 );
        PtSetResources( instance->full, 3, args );
        PtSetResource( instance->full_widget, Pt_ARG_FLAGS, 0, Pt_GHOST|Pt_BLOCKED );
    }
}

static int plugin_full_resize( PtWidget_t *widget, void *data,
    PtCallbackInfo_t *cbinfo )
{
   return Pt_CONTINUE;
}

static int plugin_full_changed( PtWidget_t *widget, void *data,
    PtCallbackInfo_t *cbinfo )
{
PluginFullInstance_t *instance = ( PluginFullInstance_t * ) data;
char *p;
PtGetResource( widget, Pt_ARG_TEXT_STRING, &p, 0 );
set_state( instance, p );
return Pt_CONTINUE;
}

static void get_value_and_apply( PluginFullInstance_t *instance ) {
char *p;
PtGetResource( instance->full_widget, Pt_ARG_TEXT_STRING, &p, 0 );
instance->n_master = strlen( p );
instance->value_master = strdup( p );
if( instance->exp->common.apply( instance->phab, instance->n_master,
instance->value_master ) ) {
    /* we matched the default */
    free( instance->value_master );
    instance->value_master = instance->default_value;
    instance->n_master = instance->n_default;
}
}

static void set_state( PluginFullInstance_t *instance, char *value ) {
if( !strcmp( value, instance->default_value ) ) {
    if( strcmp( value, instance->value_master ) )
        instance->exp->set_state( instance->convenience_handle,
RESPLUGIN_STATE_MATCH_DEFAULT );
    else
        instance->exp->set_state( instance->convenience_handle,
RESPLUGIN_STATE_MATCH_DEFAULT_MASTER );
}
else {
    if( strcmp( value, instance->value_master ) )
        instance->exp->set_state( instance->convenience_handle,
RESPLUGIN_STATE_NO_MATCH );
    else
        instance->exp->set_state( instance->convenience_handle,
RESPLUGIN_STATE_MATCH_MASTER );
}
}

static int get_value_and_apply( PluginFullInstance_t *instance ) {
    char *p;
PtGetResource( instance->full_widget, Pt_ARG_TEXT_STRING, &p, 0 );
    instance->n_master = strlen( p );
    instance->value_master = strdup( p );
    if( instance->exp->common.apply( instance->phab, instance->n_master,
instance->value_master ) ) {
        /* we matched the default */
        free( instance->value_master );
        instance->value_master = instance->default_value;
        instance->n_master = instance->n_default;
    }
}

static int plugin_full_done( PtWidget_t *widget, void *data,
PtCallbackInfo_t *cbinfo ) {
    PluginFullInstance_t *instance = ( PluginFullInstance_t * ) data;
    get_value_and_apply( instance );
    if( instance->exp->closing( instance->phab ) ) {
instance->exp->destroy( instance->convenience_handle );
free( instance );
    }
    return Pt_CONTINUE;
}

static int plugin_full_notify( ResPluginAction_t action, void *notify_data ) {
    PluginFullInstance_t *instance = ( PluginFullInstance_t * ) notify_data;
    switch( action ) {
    case RESPLUGIN_ACTION_APPLY: {
        get_value_and_apply( instance );
        set_state( instance, instance->value_master );
    }
    break;
    case RESPLUGIN_ACTION_CLOSE: {
        if( instance->exp->closing( instance->phab ) ) {
instance->exp->destroy( instance->convenience_handle );
free( instance );
    }
    break;
    case RESPLUGIN_ACTION_DEFAULT: {
        PtSetResource( instance->full_widget, Pt_ARG_TEXT_STRING,
instance->default_value, 0 );
        set_state( instance, instance->default_value );
    }
    break;
    }
}

static void plugin_full_notify( ResPluginAction_t action, void *notify_data ) {
    PluginFullInstance_t *instance = ( PluginFullInstance_t * ) notify_data;
    switch( action ) {
    case RESPLUGIN_ACTION_APPLY: {
        get_value_and_apply( instance );
        set_state( instance, instance->value_master );
    }
    break;
    case RESPLUGIN_ACTION_CLOSE: {
        if( instance->exp->closing( instance->phab ) ) {
instance->exp->destroy( instance->convenience_handle );
free( instance );
    }
    break;
    case RESPLUGIN_ACTION_DEFAULT: {
        PtSetResource( instance->full_widget, Pt_ARG_TEXT_STRING,
instance->default_value, 0 );
        set_state( instance, instance->default_value );
    }
    break;
    }
}
An external editor

The following is an example of using an external application as a resource editor in PhAB. We will use ped to edit the Pt_ARG_TEXT_STRING resource for the PtButton widget.

To use this example, compile the two files listed below, external_multi.h and external_multi.c, into a DLL named external_multi.so. Copy the external_multi.so file into the plugins/resource directory below the location that contains PhAB’s executable (usually /usr/photon/appbuilder/plugins/resource). Edit the res_editors.def in the PhAB executable directory to contain the lines:

```
e=external_multi
p=
F=multi@external_multi.so
```

Edit the file def_res.def in the same directory to contain the line:

```
external_multi
```

Edit the ptpalette.pal file in the same directory to specify external_multi as the resource editor for Pt_ARG_TEXT_STRING for the PtButton widget. To do this, in the PtButton section of the file, change the Pt_ARG_TEXT_STRING line to:

```
r=Pt_ARG_TEXT_STRING,Button Text,3011,3002,0,multi/external_multi,NULL
```

Code listing for external_multi.h

```
#ifndef __ALREADY_INCLUDED_H
#define __ALREADY_INCLUDED_H

#include <res_plugin_api.h>
#include <aoi/aoi.h>

#include <pthread.h>

typedef struct {
    int n_master;
    void *value_master;
    int n_default;
    void *default_value;
    PhABHandle_t phab;
    PhABResExportFull_t *exp;
    int pid, phab_waiting;
    char *path;
    pthread_t monitor;
} PluginFullInstance_t;

#endif
```
Code listing for `external_multi.c`

```c
#include <Pt.h>
#include <stdio.h>
#include <sys/stat.h>
#include <sys/netmgr.h>
#include <time.h>
#include <signal.h>

/* Local headers */
#include "external_multi.h"

static int plugin_loading( const char *path );
static void plugin_unloading( void );
static void plugin_full_set_data( ResPluginHandle_t handle, int n, const void *value,
    const ResPluginFormatData_t *format );
static void plugin_full_destroy( ResPluginHandle_t handle );
static ResPluginHandle_t plugin_full_create
    ( PhABHandle_t phab , const PhABResExportFull_t *exp,
    const ResPluginFormatData_t *format,
    int n_default, const void *default_value,
    PhArea_t *area, char *caption,
    int n, const void *value );
static void plugin_full_disable( ResPluginHandle_t handle );
static void plugin_full_block( ResPluginHandle_t handle, int block );
static void plugin_full_to_front( ResPluginHandle_t handle );
static int plugin_full_any_changes( ResPluginHandle_t handle );
static void plugin_full_get_changes( ResPluginHandle_t handle, int *pn, void **pvalue );
static void plugin_full_get_area( ResPluginHandle_t handle, PhArea_t *area );

static void child_exit( int sig );
static void *monitor_f( void * data );
static void get_changes_from_file( PluginFullInstance_t *instance, int *pn,
    char **value );
static void apply_from_file( PluginFullInstance_t *instance );
static void emit_wm_event( PluginFullInstance_t *instance, unsigned long event_type );
static void was_file_changed( PluginFullInstance_t *instance, int *modification_time );

static ResPluginFullEditor_t full_editors[1] =
    {
    { "multi",
    RES_DATATYPE_MULTI,
    RESPLUGIN_DATATYPE_VERSION,
    plugin_full_set_data,
    plugin_full_destroy },
    plugin_full_create,
    plugin_full_disable,
    plugin_full_block,
    plugin_full_to_front,
    plugin_full_any_changes,
    plugin_full_get_changes,
    plugin_full_get_area
    }
};

static ResPlugin_t tab = { plugin_loading,
    plugin_unloading,
    1,
    full_editors,
    0,
    NULL
};

AOInterface_t interfaces[] = {
    { "PhAB Resource Editors", RESPLUGIN_VERSION, &tab },
    { 0, 0, 0 }
};

static int plugin_loading( const char *path ) {
    signal( SIGCHLD, child_exit );
    return 0;
}
```
static void plugin_unloading( void ) {
}

static ResPluginHandle_t plugin_full_create
( PhABHandle_t phab, const PhABResExportFull_t *exp,
const ResPluginFormatData_t *format,
int n_default, const void *default_value,
PhArea_t *area, char *caption,
in_t, const void *value )
{
    PluginFullInstance_t *instance;
    FILE *fp;
    char path[PATH_MAX];
    time_t t;
    char *argv[3];

    time( &t );
    /* put the data in a file */
    sprintf( path, "/tmp/%ld.txt", (long) t );
    fp = fopen( path, "w" );
    if( !fp ) return NULL;
    if( value ) fwrite( (char*)value, 1, n, fp );
    fclose( fp );
    instance = calloc( 1, sizeof( *instance ) );
    if( !instance ) return NULL;
    instance->n_default = n_default;
    instance->default_value = default_value;
    instance->n_master = n;
    instance->value_master = value;
    instance->phab = phab;
    instance->exp = exp;
    argv[0] = "ped";
    argv[1] = path;
    argv[2] = NULL;
    instance->pid = spawnp( argv[0], 0, NULL, NULL, argv, NULL );
    if( instance->pid == -1 ) {
        free( instance );
        unlink( path );
        return NULL;
    }
    instance->path = strdup( path );
    pthread_create( &instance->monitor, NULL, monitor_f, (void*) instance );
    return ( ResPluginHandle_t ) instance;
}

static void plugin_full_destroy( ResPluginHandle_t handle )
{
    PluginFullInstance_t *instance = ( PluginFullInstance_t * ) handle;
    if( instance->pid != -1 ) {
        /* the child process is still alive, but the data inside has been asked
         * for already */
        kill( instance->pid, SIGTERM );
    }
    pthread_cancel( instance->monitor );
    pthread_detach( instance->monitor );
    unlink( instance->path );
    free( instance->path );
    free( instance );
}

static void plugin_full_disable( ResPluginHandle_t handle )
{
    PluginFullInstance_t *instance = ( PluginFullInstance_t * ) handle;
}
/* the changes inside the editor has already been taken into account */
/* close the external application if still open */
/* we need to do this because we cannot put a new data into the external application (ped), without re-spawning it */
if (instance->pid != -1) {
    kill(instance->pid, SIGTERM);
    instance->pid = -1;
    pthread_cancel(instance->monitor);
    pthread_detach(instance->monitor);
}
/* we have a disabled instance (the instance pointer still valid),
but the pid and monitor have been destroyed */

static void plugin_full_block(ResPluginHandle_t handle, int block)
{
}

static int plugin_full_any_changes(ResPluginHandle_t handle)
{
    PluginFullInstance_t *instance = (PluginFullInstance_t *)handle;
    if (instance->pid == -1) return RESPLUGIN_NO_CHANGES;
    else {
        emit_wm_event(instance, Ph_WM_TOFRONT);
        emit_wm_event(instance, Ph_WM_CLOSE);
        instance->phab_waiting = 1;
        return RESPLUGIN_WAIT;
    }
}

static void plugin_full_get_changes(ResPluginHandle_t handle, int *pn, void **pvalue)
{
    PluginFullInstance_t *instance = (PluginFullInstance_t *)handle;
    /* the changes are in the instance->path */
    get_changes_from_file(instance, pn, (char **)pvalue);
}

static void plugin_full_get_area(ResPluginHandle_t handle, PhArea_t *area)
{
    /* not used, let the external editor memorize/restore its own area */
}

static void plugin_full_to_front(ResPluginHandle_t handle)
{
    PluginFullInstance_t *instance = (PluginFullInstance_t *)handle;
    emit_wm_event(instance, Ph_WM_TOFRONT);
}

static void plugin_full_set_data(ResPluginHandle_t handle, int n, const void *value,
                                const ResPluginFormatData_t *format)
{
    PluginFullInstance_t *instance = (PluginFullInstance_t *)handle;
    FILE *fp;
    instance->n_master = n;
    instance->value_master = value;
    /* re-spawn the external application and the monitor thread */
    fp = fopen(instance->path, "w");
    if (value) fwrite((char*)value, 1, n, fp);
    fclose(fp);
    instance->pid = spawnl(P_NOWAIT, "ped", "ped", instance->path, NULL);
    pthread_create(&instance->monitor, NULL, monitor_f, (void*)instance);
}

static void child_exit(int sig)
{
    int status;
    wait(&status);
}
static void *monitor_f( void * data ) {
    PluginFullInstance_t *instance = ( PluginFullInstance_t * ) data;
    int mod = 0;
    for( ; ; ) {
        /* check if instance->pid still exists */
        if( kill( instance->pid, 0 ) == -1 ) {
            /* the child has exited */
            instance->pid = -1;
            if( instance->phab_waiting ) {
                int n, answer;
                char *value;
                get_changes_from_file( instance, &n, &value );
                if( n == instance->n_master && !memcmp( value, instance->value_master, n ) )
                    answer = RESPLUGIN_NO_CHANGES;
                else answer = RESPLUGIN_CHANGES;
                PtEnter( Pt_EVENT_PROCESS_ALLOW );
                instance->exp->answer_changes( instance->phab, answer, n, value );
                PtLeave( Pt_EVENT_PROCESS_ALLOW );
            } else {
                /* check the modification time on the file again,
                 * because maybe the user did a Exit->Save or not->Save */
                was_file_changed( instance, &mod );
                PtEnter( Pt_EVENT_PROCESS_ALLOW );
                instance->exp->closing( instance->phab );
                PtLeave( Pt_EVENT_PROCESS_ALLOW );
            }
            unlink( instance->path );
            free( instance->path );
            free( instance );
            pthread_detach( pthread_self() );
            return NULL;
        }
    }
}

static void was_file_changed( PluginFullInstance_t *instance, int *modification_time ) {
    struct stat st;
    if( stat( instance->path, &st ) == 0 ) {
        if( *modification_time != st.st_mtime ) {
            if( *modification_time ) {
                /* the file was changed - call into phab with the apply method */
                apply_from_file( instance );
            }
            *modification_time = st.st_mtime;
        }
    }
}

static void get_changes_from_file( PluginFullInstance_t *instance, int *pn, char **value ) {
    FILE *fp;
    int n;
    char *v;
fp = fopen( instance->path, "r" );
if( !fp ) return;

fseek( fp, 0, SEEK_END );
n = ftell( fp );

free( fp );
fclose( fp );

*pn = n;
*vvalue = v;
}

static void apply_from_file( PluginFullInstance_t *instance ) {
int n;
char *values;

get_changes_from_file( instance, &n, &values );
instance->value_master = values;

PtEnter( Pt_EVENT_PROCESS_ALLOW );
if( instance->exp->common.apply( instance->phab, instance->n_master, instance->value_master ) ) {
  /* we matched the default */
  free( instance->value_master );
  instance->value_master = instance->default_value;
  instance->n_master = instance->n_default;
}

PtLeave( Pt_EVENT_PROCESS_ALLOW );
}

static PhConnectId_t get_connect_id( pid_t pid )
{

PhConnectInfo_t buf;
PhConnectId_t id = 0;

while ( (!id = PhGetConnectInfo(id, &buf)) != -1 
  && (buf.pid != pid ||
    ND_NODE_CMP(buf.nid, ND_LOCAL_NODE)))
  ++id;

return id;
}

static void emit_wm_event( PluginFullInstance_t *instance, unsigned long event_type )
{

PhWindowEvent_t event;
PhConnectId_t connection_id = get_connect_id( instance->pid );

memset( &event, 0, sizeof( event ) );
event.event_f = event_type;
PtForwardWindowTaskEvent( connection_id, &event );
}
Appendix A

Miscellaneous Widget-Building Tips
Here are some general widget-building tips (in no particular order):

- Never call `PgSetRegion()`.
- Never call `PtDamageWidget()` from a Draw method.
- Never call `Pg...` functions from anywhere except the Draw method.
- You don’t need to call `PtDamageWidget()` after calling `PtRealizeWidget()`.
- In the modify functions, don’t call the widget’s `Extent` method directly. Instead, set `Pt_WIDGET_RESIZE` in the widget’s `Pt_ARG_FLAGS` resource; the widget library will calculate the widget’s extent and perform any visual cleanup at the appropriate time.
- If you must calculate the extent immediately, call `PtExtentWidget()`, not your widget’s `Extent` method.
- Use `PtSetResources()` to set the resources of subordinate (procreated) widgets.
- Avoid calling `PgFlush()` — the library will call it automatically when required.
- Never call `PtBkgdHandlerProcess()` from a Draw method.
- It’s generally best to create subordinate children in your `Defaults` method; this way they can be modified as needed via resources before being realized. Another upside to this is that the widgets exist at the time the parent is being realized.

When creating the widgets in methods from `Initialization` to `Connect`, the widgets don’t exist at the time of realization and may be skipped. In this case, you need to have a `Realized` method in which you realize the children you want visible.

It’s also important to have an `Unrealized` method in which you destroy the subordinates or you must check to make sure you don’t already have the subordinates in your `Initialization` method. Failing to do either of these will likely produce a memory leak.

- If you have an `Extent` method, it must actually calculate the extent of the widget. It’s a fairly common mistake to do lots of dimension, layout, and area calculations and then forget to actually set the widget’s extent. Setting the extent is usually accomplished via calling:

  ```c
  PtSuperClassExtent( PtBasic "or another superclass", widget);
  ```

- If your widget’s subordinate children aren’t being displayed, check the widget’s `resize` policy (`Pt_ARG_RESIZE_FLAGS`). This resource indicates whether or not the widget’s size is adjusted based on the size and location of its children.

- If you have a `Draw` method defined, make sure you call `PtClipRemove()` if you called `PtClipAdd()` earlier in the function. Don’t adjust the draw offset via `PgSetTranslation()` because the translation is already set up before your `Draw` function is called. If you really do want to change the translation, change it back before exiting your `Draw` method.
accelerator

See hotkey.

activate

A widget is usually activated when you release a mouse button while pointing at an armed widget.

active window

The window that currently has focus.

anchor offset

The distance between the edges of a widget and the parent widget it’s anchored to.

anchor

A constraint mechanism used to manage what happens to a widget when its parent is expanded or contracted. For example, a pane that’s anchored to the sides of a window expands or contracts as the window’s size is changed.

application region

A region that belongs to a Photon application (as opposed to a Photon system process, such as the window manager, graphics drivers, etc.). An application region is usually placed behind the device region. Also called a window region.

argument list

An array of type PtArg_t used when setting and getting widget resources.

arm

A widget is usually armed when you press a mouse button while pointing at it.

backdrop

An image that’s displayed as a background on your screen.

backdrop region

A region placed behind all windows to display a background image.

balloon

A small box that pops up to define or explain part of the user interface. A balloon is displayed when the pointer pauses over a widget.

bitmap

A color picture consisting of one or more bitplanes.
bitplane

An array of bits representing pixels of a single color in a bitmap.

blit

An operation that moves an area of a graphics context (e.g. the screen) to another area on the same or a different context.

callback

A callback function or a callback resource.

callback function

Code connecting an application’s user interface to its code. For example, a callback is invoked when you press a button.

callback resource

A resource that specifies a list of functions and their client data to be called when a certain action occurs.

canvas

The part of a widget that’s used for drawing. For PtWidget, this is the area inside the widget’s borders. For PtBasic and its descendants, the canvas is the area inside the widget’s border and margins. Other widgets, such as PtLabel, may define additional margins.

class

See widget class.

class hierarchy

The relationships between all of the widget classes.

client data

Any arbitrary data the application may need to provide to a callback function.

clipping list

An array of rectangles used to restrict output to a particular area.

clipping rectangle

A rectangle used to restrict output to a particular area.

CMY value

A color expressed as levels of cyan, magenta, and yellow.
CMYK value

A color expressed as levels of cyan, magenta, yellow, and black.

code-type link callback

In a PhAB application, an application function that’s called when a widget’s callback list is invoked.

color depth

The number of bits per pixel for a screen or pixmap.

Common User Access

See CUA.

compose sequence

A sequence of key presses that can be used to type a character that might not appear on the keyboard.

console

One of nine virtual screens on the desktop. Also called a workspace.

consume

When a widget has processed an event and prevents another widget from interacting with the event, the first widget is said to have consumed the event.

container

A widget that can have other widgets as children. For example, PtWindow, PtGroup, and PtOSContainer.

cooked event

A key or pointer event that has been assigned a location in the Photon event space. Also called a focused event.

CUA

Common User Access — a standard that defines how you can change focus by using the keyboard.

current item

The item in a list or tree widget that will be selected (or perhaps unselected) when you press Enter or Space. It’s typically drawn with a blue dotted line around it when its widget has focus.
cursor

An indicator of a position on a screen, such as a pointer or an insertion point in a text field.

damaged

Whenever a widget needs to be redisplayed due to a change in the window (e.g. the widget is changed, moved, or realized), it’s said to be damaged.

dead key

A key that, when pressed, doesn’t produce a symbol, but initiates a compose sequence.

default placement

The placement of a region when no siblings are specified. The opposite of specific placement.

desktop

The virtual screen, consisting of nine consoles or workspaces.

device region

The region located in the middle of the event space, with application regions behind it and driver regions in front of it (from the user’s point of view).

dialog module

A PhAB module similar to a window module, except that a dialog module can have only one instance per process.

direct-color

A color scheme in which each pixel is represented by an RGB value. Contrast palette-based.

disjoint parent

A disjoint widget that’s the ancestor of another widget.

disjoint widget

A widget that can exist without a parent. If a disjoint widget has a parent, it can exist outside its parent’s canvas. For example, PtWindow, PtMenu, and PtRegion are disjoint widgets, but PtButton, PtBkgd, and PtRect aren’t.

A disjoint widget owns regions that aren’t children of its parent’s regions. Any clipping set by the parent of a disjoint widget isn’t applied to the disjoint widget. The regions of disjoint widgets are sensitive and opaque to expose events.
**dithering**
A process whereby pixels of two colors are combined to create a texture or a blended color.

**draw context**
A structure that defines the flow of the draw stream. The default draw context emits draw events to graphics drivers. **Print contexts** and **memory contexts** are types of draw contexts.

**draw stream**
A series of tokens that are dispatched via draw events and can be collected by a rendering engine such as a graphics driver.

**driver region**
A **region** created by a driver, usually placed in front of the **device region**.

**encapsulation driver**
A program that displays Photon graphical output inside another windowing system such as the X Window System.

**event**
A data structure that represents an interaction between you and an application or between applications. Events travel through the event space either toward you or away (i.e. toward the **root region**).

**event compression**
The merging of events such that the application sees only their latest values. The application doesn’t have to process many unnecessary events.

**event handler**
A callback function that lets an application respond directly to Photon events, such as dragging events.

**event mask**
A set of event types that are of interest to an **event handler**. When one of these events occurs, the event handler is invoked.

**event space**
An abstract, three-dimensional space that contains regions — from the root region at the back to the graphics region at the front. You sit outside the event space, looking in from the front. Events travel through the event space either toward the root region or toward you.
exported subordinate child

A widget created by a container widget (as opposed to an application) whose resources you can access only through the parent.

exposure

Typically occurs when a region is destroyed, resized, or moved. Expose events are sent to applications to inform them when the contents of their regions need to be redisplayed.

extent

A rectangle that describes the outermost edges of a widget.

File Manager

The Photon File Manager (PFM), an application used to maintain and organize files and directories.

focus

A widget that has focus will receive any key events collected by its window.

focus region

A region placed just behind the device region by the Photon Window Manager that lets it intercept key events and direct them to the active window.

focused event

A key or pointer event that has been assigned a location in the Photon event space. Also called a cooked event.

folder

In the Photon File Manager, a metaphor for a directory.

GC

See graphics context.

gameometry negotiation

The process of determining the layout for a widget and its descendants, which depends on the widget’s layout policy, any size set for the widget, and the dimensions and desired positions of each of the widget’s children.

global header file

A header file that’s included in all code generated by PhAB for an application. The global header file is specified in PhAB’s Application Startup Information dialog.
graphics driver
A program that places a region that’s sensitive to draw events on the user’s side of the device region, collects draw events, and renders the graphical information on the screen.

graphics context (GC)
A data structure that defines the characteristics of primitives, including foreground color, background color, line width, clipping, etc.

Helpviewer
A Photon application for viewing online documentation.

hotkey
A special key or keychord that invokes an action (such as a menu item) without actually selecting a widget. Also called an accelerator. Contrast keyboard shortcut.

hotspot
The part of the pointer that corresponds to the coordinates reported for the pointer (e.g. the intersection of crosshairs, or the tip of the arrow of the basic pointer).

HSB
Hue-Saturation-Brightness color model.

HSV
Hue-Saturation-Value color model.

icon module
A PhAB module that associates icons with an application.

image
A rectangular array of color values, where each element represents a single pixel. See also direct-color and palette-based.

initialization function
In a PhAB application, a function that’s called before any widgets are created.

input driver
A program that emits, and is the source of, key and/or pointer events.

input group
A set of input and output devices. There’s typically one input group per user.
input handler (or input-handling function)

A function that’s hooked into Photon’s main event-processing loop to handle messages and pulses sent to the application by other processes.

instance

A concrete example of an abstract class; for example, “Lassie” is an instance of the class “dog.” In Photon, an instance is usually a widget instance; for example, a pushbutton is an instance of the PtButton widget class. When an instance of a widget is created, the initial values of its resources are assigned.

instance name

In PhAB, a string that identifies a particular instance of a widget so that you can access the instance in your application’s code.

instantiation

The action of creating an instance of a widget class in an application.

internal link

A PhAB mechanism that lets a developer access a PhAB module directly from an application’s code.

Image Viewer

A Photon application (pv) that displays images.

key modifier

A flag in a key event that indicates the state of the corresponding modifier key when another key was pressed.

keyboard driver

A program that gets information from the keyboard hardware, builds Photon key events, and emits them towards the root region.

keyboard shortcut

A key that selects a menu item. The shortcut works only if the menu is displayed. Contrast hotkey.

language database

A file that contains the text strings used in a PhAB application; a language database makes it easier to create multilingual applications with PhAB’s language editor.

link callback

A mechanism that connects different parts of a PhAB application. For example, a link callback can be invoked to display a dialog when a button is pressed.
margin

The area between a widget’s border and canvas.

memory context

A draw context in which Photon draw events are directed to memory for future displaying on the screen, as opposed to a printer (print context) or to the screen directly (the default draw context).

menu module

A PhAB module used to create a menu.

method

A function that’s internal to a widget class and invoked under specific conditions (e.g. to draw the widget). Methods are provided as pointers to functions in widget class records.

modifier key

A key (such as Shift, Alt, or Ctrl) used to change the meaning of another key.

module

An object in PhAB that holds an application’s widgets. PhAB modules include windows, menus, icons, pictures, and dialogs.

module-type link callback

A link callback that displays a PhAB module.

mouse driver

A program that gets information from the pointer hardware, builds Photon raw pointer events, and emits them towards the root region.

opaque

The state of a region with regard to events. If a region is opaque to an event type, any event of that type that intersects with the region has its rectangle set adjusted to clip out the intersecting area. The region prevents the event from passing through.

palette

An array of colors. A hard palette is in hardware; a soft palette is in software.

palette-based

A color scheme in which each pixel is represented by an index into a palette. Contrast direct-color.
PDR


PFM

See Photon File Manager.

PhAB

Photon Application Builder. Visual design tool that generates the code required to implement a user interface.

phditto

A utility that accesses the Photon workspace on a remote node. See also ditto.

Phindows

Photon in Windows. An application that accesses a Photon session from a Microsoft Windows environment.

Photon File Manager (PFM)

An application used to maintain and organize files and directories.

Photon Manager or server

The program that maintains the Photon event space by managing regions and events.

Photon Terminal

An application (pterm) that emulates a character-mode terminal in a Photon window.

Photon Window Manager (PWM)

An application that manages the appearance of window frames and other objects on the screen. For example, the window manager adds the resize bars, title bar, and various buttons to an application’s window. The window manager also provides a method of focusing keyboard events.

picture module

A PhAB module that contains an arrangement of widgets that can be displayed in another widget or used as a widget database.

pixmap

A bitmap or image.

plane mask

A mask used to restrict graphics operations to affect only a subset of color bits.
point source

A single-point rectangle set used as the source of an event.

pointer

An object on the screen that tracks the position of a pointing device (e.g. a mouse, tablet, track-ball, or joystick). Photon has several pointers indicating various states: Basic, Busy, Help, Move, Resize, I-beam, No-input.

Press-drag-release (PDR)

A method of selecting a menu item by pressing down a mouse button while pointing to a menu button, dragging until the desired item is highlighted, and releasing the mouse button.

print context

A draw context in which Photon draw events are directed to a file, as opposed to the screen (the default draw context) or to memory (memory context).

printer driver

A program that converts Photon draw stream format into a format suitable for a printer, including PostScript, Hewlett-Packard PCL, and Canon.

procreated widget

A widget created by another widget (as opposed to an application), such as the PtList and PtText created by a PtComboBox. Also known as a subordinate child.

pterm

A Photon Terminal; an application that emulates a character-mode terminal in a Photon window.

pulse

A small message that doesn’t require a reply; used for asynchronous communication with a Photon application.

pv

See Image Viewer.

PWM

See Photon Window Manager.

raw event

An input event that hasn’t been assigned a location in the Photon event space. Also called an unfocused event.
raw callback

A function that lets an application respond directly to Photon events such as dragging events. Also called an event handler.

realize

To display a widget and its descendants, possibly making them interactive.

rectangle set

An array of nonoverlapping rectangles associated with an event.

region

A rectangular area within the Photon event space that’s used by an application for collecting and emitting events.

resize policy

A rule that governs how a widget resizes itself when its contents change.

resource

An attribute of a widget, such as fill color, dimensions, or a callback list.

root region

The region at the very back of the Photon event space.

sensitive

The state of a region with regard to events. If a region is sensitive to a particular type of event, the region’s owner collects a copy of any such event that intersects with the region.

setup function

A function that’s called after a PhAB module is created.

shelf

An application that attaches areas to the outside edge of the screen. You can add plugins to customize these areas, such as a taskbar, launcher, clock, and magnifier.

Snapshot

A Photon application for capturing images of the screen.

specific placement

The placement of a region when one or more siblings are specified. The opposite of default placement.
**subordinate child**

A widget created by another widget (as opposed to an application), such as the `PtList` and `PtText` created by a `PtComboBox`. Also known as a **procreated widget**.

**table-of-contents (TOC) file**

In the Photon **Helpviewer**, a file that describes a hierarchy of help topics.

**taskbar**

A shelf plugin that displays icons representing the applications that are currently running.

**tile**

A data structure used to build linked lists of rectangles, such as a list of the damaged parts of an interface.

**topic path**

Help information identified by a string of **titles** that are separated by slashes.

**topic root**

A topic path that’s used as a starting point for locating help topics.

**topic tree**

A hierarchy of help information.

**translation file**

A file containing translated strings for a PhAB application. There’s one translation file per language supported by the application.

**unfocused event**

See **raw event**.

**Unicode**

The ISO/IEC 10646 16-bit encoding scheme for representing the characters used in most languages.

**UTF-8**

The encoding for **Unicode** characters, where each character is represented by one, two, or three bytes.

**widget**

A component (e.g. a pushbutton) in a graphical user interface.
widget class

A template for widgets that perform similar functions and provide the same public interface. For example, PtButton is a widget class.

widget database

In PhAB, a module containing widgets that can be copied at any time into a window, dialog, or other container.

widget family

A hierarchy of widget instances. For example, a window and the widgets it contains.

widget instance

See instance.

window frame region

A region that PWM adds to a window. It lets you move, resize, iconify, and close the window.

Window Manager

See Photon Window Manager.

window module

A PhAB module that’s instantiated as a PtWindow widget.

window region

A region that belongs to an application window.

work procedure

A function that’s invoked when there are no Photon events pending for an application.

workspace

See console.

workspace menu

A configurable menu that’s displayed when you press or click the right mouse button while pointing at the background of the desktop.
Index

A

allocating
  image resources  252
anchors  9, 13
  container widgets  83
    invalidating  86
Extent method  61, 83
PtAnchorDeregister()  163
PtAnchorRegister()  164
PtAnchorWidget()  165
PtApplyAnchors()  166
PtCalcAnchorOffsets()  169
PtSetExtentFromArea()  196
PtSuperClassExtent()  214
attributes  See resources

B

basic widgets  9, See also PtBasic
behavior
  defining in widget class  4
  inheriting from superclass  4
blocking input
  in PhAB plugins  278

calls

C

Calc Opaque Rect method  53, 200
inheriting  18
PtBasic  41, 53, 79
PtCompound  89
PtContainer  84
PtGauge  106
PtGenList  92
PtGenTree  98
PtGraphic  104
PtLabel  101
callbacks  4, 147
  balloon  92, 101, 102
class-level  See methods
common  54
describing in PhAB  141, 146
freeing  179
hotkey  73, 74, 77, 147
  freeing  180
inheriting  147
invoking  186, 189, 190
Pt_ARG_IS_CALLBACK  30
Pt_ARG_IS_CALLBACK_LIST  30, 32, 74
Pt_CB_ACTIVATE  54, 80
Pt_CB_ARM  54, 79
Pt_CB_DISARM  54, 79, 80
Pt_CB_DIVIDER_DRAG  93
Pt_CB_FILTER  74
Pt_CB_GEN_TREE_INPUT  99
Pt_CB_GOT_FOCUS  79
Pt_CB_LOST_FOCUS  79
Pt_CB_MENU  79
Pt_CB_REALIZED  48
Pt_CB_REPEAT  79
Pt_CB_RESCALE  105
Pt_CB_SCROLL_MOVE  96
Pt_CB_SCROLLED_X  66
Pt_CB_UNREALIZED  74
raw  37, 39, 54, 55, 80, 87, 94, 147, 220
  freeing  181

September 10, 2007
PtBasic 79
PtCompound 89
PtContainer 85
PtGauge 106
PtGenTree 99
PtGraphic 104
PtLabel 101
PtWidget 74
resize 83
invoking 191
canvas
resizing 194
canvas, calculating 171
chaining See methods
changes
applying
PhAB plugins 255
child redirector function 9
child-constraint methods 22
Child Created 57, 201
PtContainer 84
PtGenList 92
PtGenTree 98
PtGroup 57
Child Destroyed 58, 202
PtContainer 85
PtGenList 93
PtGenTree 98
Child Getting Focus 59, 203
PtContainer 85
PtGenList 93
PtGenTree 98
Child Getting Resources 59, 204
PtContainer 85
PtGenList 93
PtGenTree 98
Child Losing Focus 59, 205
PtContainer 85
PtGenList 93
PtGenTree 99
Child Moved Resized 206
Child Moved/Resized 58
PtContainer 84
PtGenList 93
PtGenTree 98
Child Realized 58, 208
PtContainer 84
PtGenList 92
PtGenTree 98
Child Realizing 58
Child Redirection 59
Child Setting Resources 58, 209
PtContainer 85
PtGenList 93
PtGenTree 98
Child Unrealized 58, 210
PtContainer 84
PtGenList 93
PtGenTree 98
Child Unrealizing 58
child-redirector function 22, 59, 66
PtMenuBar 59
children
anchoring 9, 83
blocking resources 9
bounding box 171
redirecting parentage 10
class
creation function 14
constraint methods 60
example 35
exporting subordinate widgets 23, 67
initializing class structure 6
PtComboBox 63
PtTimer 55
setting up event-handling 4
ShadowedBox 14, 176
defining behavior of 4
hierarchy 9
pointer 6
resource table 39
structure
example 6–8
initializing 6
class-level callbacks See methods
clipping 46, 48, 49, 81, 167, 194
balloon 102
disjoint widgets 39
PtClipAdd() 20, 172
PtClipRemove() 20, 173
PtGenList 111
closing
<table>
<thead>
<tr>
<th>Class</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>PtContainer</td>
<td>84</td>
</tr>
<tr>
<td>PtGauge</td>
<td>105</td>
</tr>
<tr>
<td>PtGenList</td>
<td>92</td>
</tr>
<tr>
<td>PtGenTree</td>
<td>97</td>
</tr>
<tr>
<td>PtGraphic</td>
<td>103</td>
</tr>
<tr>
<td>PtLabel</td>
<td>100</td>
</tr>
<tr>
<td>ShadowedBox</td>
<td>18</td>
</tr>
</tbody>
</table>

**F**

- **focus** 79
  - Common User Access (CUA) 61
  - getting 18, 22, 41, 52, 56, 59, 75, 79
  - preventing 61
  - PtGenList 92
  - losing 18, 22, 41, 52, 56, 59, 79
  - PtGenList 92
  - Pt_DIRECTED_FOCUS 75
  - Pt_EV_KEY 74
  - Pt_CHILD_GETTING_FOCUS 59
  - Pt_CHILD_LOSSING_FOCUS 59
  - Pt_DAMAGE_ON_FOCUS 52
  - Pt_FOCUS_RENTER 92
  - Pt_GEN_LIST_NO_AUTOFOCUS 111
  - Pt_GETS_FOCUS 75, 96
  - Pt_LIST_SCROLLBAR_GETS_FOCUS 96
  - Pt_SELECTION_MODE_NOFOCUS 92, 112
  - PtBasic 3
  - PtSuperClassChildGettingFocus() 203
  - PtSuperClassChildLosingFocus() 205
  - PtSuperClassGotFocus() 217
  - PtSuperClassLostFocus() 52, 219

- **freed**
  - images in PhAB plugins 264
  - memory in PhAB plugins 265

- **frugal editor**
  - functions 273
  - base structure 289
  - creating 283
structure 285
full editor functions 274
full editors creating 286
window area 291

G

Get Resources method 38, 39, 51
inheriting 17
PtCompound 69
PtFindResource() 183
Got Focus method 52, 217
inheriting 18
PtBasic 41, 79
PtButton 52
PtCompound 89
PtContainer 84
PtGauge 106
PtGenList 98
PtGenTree 97
PtGraphic 103
PtLabel 100
PtWidget 73
ShadowedBox 18
instance structure
example 5
setting/getting members 6

L

LD_LIBRARY_PATH 140
List Attributes method
list widgets 115
PtGenList 94
List Draw method
list widgets 109, 111, 112
PtGenList 93
PtGenTree 99, 129, 130
PtSuperClassGenListDraw() 118
tree widgets 129
List Inflate method
list widgets 114
PtGenList 94
PtSuperClassGenListInflate() 119
List Key method
list widgets 109, 113
PtGenList 93
PtGenTree 99
PtSuperClassGenListKey() 121
List Mouse method
list widgets 109, 112
PtGenList 93
PtGenTree 99
PtSuperClassGenListMouse() 123
List Select method
list widgets 114
PtGenList 93
PtSuperClassGenListSelect() 125
list widgets 109–117, See also PtGenList

H

handles
in PhAB plugins 267
header file
example code 5
hotkey callbacks 73, 74, 77, 147
freeing 180

I

image resources
allocating 252
indicating changes
in PhAB plugins 276
Initialization method
chaining 17
methods
  Attributes 115
  Draw 109–111
  List Draw 109, 111, 112
  List Inflate 114
  List Key 109, 113
  List Mouse 109, 112
  List Select 114
loading
  PhAB plugins 293
Lost Focus method 52, 219
  inheriting 18
    PtBasic 41, 52, 79
    PtCompound 89
    PtContainer 84
    PtGauge 106
    PtGenTree 98
    PtGraphic 104
    PtLabel 101
M
memory
  allocating
    in PhAB plugins 253
message
  windows
    in PhAB plugins 269
methods 5, 6
  Calc Opaque Rect 53, 200
    inheriting 18
      PtBasic 41, 53, 79
      PtCompound 89
      PtContainer 84
      PtGauge 106
      PtGenTree 98
      PtGraphic 104
      PtLabel 101
chaining 16
  stopping 17
child-constraint See child-constraint methods
  Connection 16, 37, 39, 46
    chaining 17
PtBasic 78
PtCompound 88
PtContainer 83
PtGauge 105
PtGenList 92
PtGenTree 97
PtGraphic 103
PtLabel 101
PtMenu 46
PtSuperClassConnect() 211
PtSuperClassConnectFrom() 211
PtWidget 46, 48, 73
ShadowedBox 18
Defaults 37, 39, 43, 66
  chaining 17
  compound widgets 23
PtBasic 78
PtComboBox 65, 67
PtCompound 67, 88
PtContainer 60, 83
PtGauge 105
PtGenList 91
PtGenTree 97
PtGraphic 103
PtLabel 100
PtWidget 73
ShadowedBox 8, 15, 42
Destruction 21, 31, 37, 39, 51
  chaining 17
PtBasic 79
PtButton 51
PtCompound 70, 89
PtContainer 84
PtGauge 105
PtGenList 92
PtGenTree 98
PtGraphic 104
PtWidget 74
Draw 16, 39, 43, 46, 48, 167, 194, 212, 317
  inheriting 17
  list widgets 109–111
PtBasic 79
PtButton 48
PtCompound 89
PtContainer 84
PtGauge 105
Index

PtGenList  92
PtGenTree  98
PtGraphic  104
PtLabel   101
repairing damage  20
ShadowedBox  18, 42
using the Pg library safely  49
Extent  16, 21, 29, 39, 45, 192, 214, 317
inheriting  17
PtBasic  78
PtCompound  88, 91
PtContainer  61, 83, 214
PtDivider   91
PtGauge    105
PtGenList  91
PtGenTree  97
PtGraphic  103, 214
PtLabel   100, 214
PtWidget   73
ShadowedBox  18
Get Resources  38, 39, 51
inheriting  17
PtCompound  69
PtFindResource()  183
Got Focus   52, 217
inheriting  18
PtBasic  41, 79
PtButton   52
PtCompound  89
PtContainer  84
PtGauge    106
PtGenList  92
PtGenTree  98
PtGraphic  104
PtLabel   101
inheriting  16, 17
Initialization  16, 37, 39, 44
chaining  17
PtBasic  78
PtCompound  88
PtContainer  83
PtGauge    105
PtGenList  91
PtGenTree  97
PtGraphic  103
PtLabel   100
PtWidget   73
ShadowedBox  18
List Attributes
list widgets  115
PtGenList  94
List Draw
list widgets  109, 111, 112
PtGenList  93
PtGenTree  99, 129, 130
PtSuperClassGenListDraw()  118
tree widgets  129
List Inflate
list widgets  114
PtGenList  94
PtSuperClassGenListInflate()  119
List Key
list widgets  109, 113
PtGenList  93
PtGenTree  99
PtSuperClassGenListKey()  121
List Mouse
list widgets  109, 112
PtGenList  93
PtGenTree  99
PtSuperClassGenListMouse()  123
List Select
list widgets  114
PtGenList  93
PtSuperClassGenListSelect()  125
Lost Focus   52, 219
inheriting  18
PtBasic  41, 52, 79
PtCompound  89
PtContainer  84
PtGauge    106
PtGenList  92
PtGenTree  98
PtGraphic  104
PtLabel   101
PtComboBox  69
PtCompound  69, 88
PtContainer  62, 83
Index

PtGauge 105
PtGenList 92
PtGenTree 97
PtGraphic 103
PtLabel 101
PtWidget 74
ShadowedBox 18
Set Resources 39, 51
  inheriting 17
PtCompound 69
PtFindResource() 183
Tree Draw Item
PtGenTree 99
  PtSuperClassGenTreeDrawItem() 134
  tree widgets 129, 130
Tree Item State
PtGenTree 99
  PtSuperClassGenTreeItemState() 135
  tree widgets 131
Unrealization 21, 37, 39, 50
  chaining 17
PtBasic 79
PtCompound 89
PtContainer 84
PtGauge 105
PtGenList 92
PtGenTree 98
PtGraphic 104
PtLabel 50, 101
PtWidget 50, 74
ShadowedBox 21
PgSetFillXORColor() 50
PgSetMultiClip() 49
PgSetPalette() 50
PgSetPlaneMask() 49
PgSetRegion() 50, 317
PgSetStrokeCap() 49
PgSetStrokeColor() 50
PgSetStrokeDash() 50
PgSetStrokeDither() 50
PgSetStrokeJoin() 50
PgSetStrokeTransPat() 50
PgSetStrokeWidth() 50
PgSetStrokeXORColor() 50
PgSetTextColor() 50
PgSetTextTransPat() 50
PgSetTextXORColor() 50
PgSetTranslation() 49
PgSetUnderline() 49
PgSetUserClip() 49
Ph_DIRECTED_FOCUS 75
Ph_EV_BOUNDARY 54, 77, 80
Ph_EV_BUT_PRESS 54, 74, 79, 94
Ph_EV_BUT_RELEASE 54, 74, 80, 94
Ph_EV_BUT_REPEAT 54, 74, 79, 94
Ph_EVExpose 157
Ph_EV_KEY 74, 94
Ph_EV_PTR_MOTION_BUTTON 94
Ph_EXPOSE_FAMILY 157
Ph_EXPOSE_REGION 157
Ph_REGION_CURSOR 76
PhAB
  plugins
    allocating memory 253
    answering changes 254
    applying changes 255
    base structure 275
    blocking input 278
    closing 257
    common structure 271, 279
    destroying windows 261
    disabling 282
    duplicating values 262
    freeing images 264
    freeing memory 265
    frugal editor functions 273
    full editor functions 274

P

pathname delimiter in QNX Momentics documentation xvii
Pg_BITMAP_BACKFILL 101
Pg_TRANSPARENT 78
Pg... functions 317
PgFlush() 317
PgSetClipping() 49
PgSetDrawMode() 49
PgSetFillColor() 50
PgSetFillDither() 50
PgSetFillTransPat() 50

342  Index  September 10, 2007
handles 267
indicating changes 276
messages 269
opening full editor 270
releasing resources 281
window area 266
PhAB plugins
creating windows 259
freeing memory 263
frugal editors 283, 285
full editors 286, 289
window area 291
loading 293
resource editors
getting changes 292
PhAB, adding custom widgets 4, 139
Photon library
API functions 153
widget superclasses 8
PhRegionChange() 73, 76
PhRegionOpen() 73
Pt_ANCHORS_INVALID 86
Pt_ANCHORS_LOCKED 86
Pt_ARG_BALLOON_TEXT 102
Pt_ARG_DATA 67
Pt_ARG_DIM 73, 143
Pt_ARG_DIVIDER_OFFSET 92, 93, 95
Pt_ARG_DIVIDER_SIZES 93
Pt_ARG_FLAGS 53, 79, 81, 317
Pt_ARG_IS_ALLOC 30, 32, 74
Pt_ARG_IS_ARRAY 31, 33, 74
Pt_ARG_IS_BOOLEAN 31, 33
Pt_ARG_IS_CALLBACK 30
Pt_ARG_IS_CALLBACK_LIST 30, 32, 74
Pt_ARG_IS_FLAGS 30, 31
Pt_ARG_IS_IMAGE 31, 33
Pt_ARG_IS_LINK 30, 32, 74
Pt_ARG_IS_NUMBER 30, 31, 33
Pt_ARG_IS_POINTER 30, 32
Pt_ARG_IS_STRING 30, 31, 74
Pt_ARG_IS_STRUCT 30, 31
Pt_ARG_LIST_COLUMN_POS 93
Pt_ARG_LIST_FLAGS 96
Pt_ARG_POS 73, 143
Pt_ARG_TOP_ITEM_POS 110
Pt_AUTO_EXTENT 87
Pt_AUTOHIGHLIGHT 77, 79, 80
Pt_BALLOON_AS_REQUIRED 102, 103
Pt_BASIC 39
Pt_BITMAP 101
Pt_BLIT 192
Pt_BLIT_FORCE 192
Pt_BLOCK_CUA_FOCUS 61
Pt_CALLBACKS_ACTIVE 92
Pt_CB_ACTIVATE 54, 80
Pt_CB_ARM 54, 79
Pt_CB_DISARM 54, 79, 80
Pt_CB_DIVIDER_DRAG 93
Pt_CB_FILTER 74
Pt_CB_GEN_TREE_INPUT 99
Pt_CB_GOT_FOCUS 79
Pt_CB_LOST_FOCUS 79
Pt_CB_MENU 79
Pt_CB_RAW 54
Pt_CB_REALIZED 48
Pt_CB_REPEAT 79
Pt_CB_RESCALE 105
Pt_CB_SCROLL_MOVE 96
Pt_CB_SCROLLED_X 66
Pt_CB_UNREALIZED 74
Pt_CHANGE_CANVAS 29
Pt_CHANGE_CANVAS_REDRAW 30
Pt_CHANGE_INTERIOR 29
Pt_CHANGE_INVISIBLE 29
Pt_CHANGE_PREVENT 29
Pt_CHANGE_REDRAW 29
Pt_CHANGE_RESIZE 29, 45
Pt_CHANGE_RESIZE_REDRAW 29, 45
Pt_CHILD_CREATED 57
Pt_CHILD_DESTROYED 58
Pt_CHILD_GETTING_FOCUS 59
Pt_CHILD_GETTING_RESOURCE 59
Pt_CHILD_LOSING_FOCUS 59
Pt_CHILD_MOVED_RESIZED 58
Pt_CHILD_REALIZED 58
Pt_CHILD_REALIZING 58
Pt_CHILD_REDIRECTOR 59
Pt_CHILD_SETTIGN_RESOURCE 58
Pt_CHILD_UNREALIZED 58
Pt_CHILD_UNREALIZING 58
Pt_CONSUME_EVENTS 81
Pt_CONTAINER 39

© 2007, QNX Software Systems GmbH & Co. KG.

September 10, 2007

Index 343
Pt_TREE_EXPANDING 131, 135
Pt_TREE_ITEM_EXPANDABLE 129
Pt_TREE_ITEM_EXPANDED 129, 131
Pt_TREE_TO_RIGHT 100
Pt_UCLIP 46, 49, 167, 194
Pt_USER 28
Pt_WIDGET_RESIZE 102, 317
Pt_Z_STRING 100, 101
PtAddWidgetData() 162
PtAnchorDeregister() 163
PtAnchorRegister() 164
PtAnchorWidget() 13, 165
PtApplyAnchors() 13, 166
PtArc
   Pt_UCLIP 49
PtAttemptResize() 167
PtBasic 77
   additional resources 41
   methods 43
      Calc Opaque Rect 41, 53, 79
      Connection 78
      Defaults 78
      Destruction 79
      Draw 79
      Extent 78
      Got Focus 41, 52, 79
      Initialization 78
      Lost Focus 41, 52, 79
      Realization 78
      Unrealization 79
Pt_DAMAGE_ON_FOCUS 52
   subclassing 3
   widget actions 79
PtBkgdHandlerProcess() 317
PtButton
   methods
      Draw 48
      widget actions 54
PtCalcAnchorOffsets() 169
PtCalcCanvas() 45
PtCalcRegion() 170
PtChildBoundingBox() 171
PtClipAdd() 20, 49, 172
PtClippAdd() 49
PtClipRemove() 20, 49, 173
PtComboBox
class-creation function 63
exported subordinate children 23
methods
   Defaults 65, 67
   Realization 69
PtCompound 87
   methods
      Calc Opaque Rect 89
      Connection 88
      Defaults 67, 88
      Destruction 70, 89
      Draw 89
      Extent 88, 91
      Get Resources 69
      Got Focus 89
      Initialization 88
      Lost Focus 89
      Realization 69, 88
      Set Resources 69
      Unrealization 89
      widget actions 89
PtCompoundRedirect() 60, 66, 92, 174
PtContainer 56–62, 81
   initializing 44
   methods
      See also child-constraint methods
      Calc Opaque Rect 84
      Connection 83
      Defaults 60, 83
      Destruction 84
      Draw 84
      Extent 61, 83
      Got Focus 84
      Initialization 83
      Lost Focus 84
      Realization 62, 83
      Unrealization 84
      widget actions 85
PtContainerHit() 75
PtCoreChangeRegion() 175
PtCreateWidget() 13–16, 43
PtCreateWidgetClass() 15, 41, 176
PtDamageExposed() 178
PtDamageWidget() 77, 317
PtDestroyCallbackList() 179
PtDestroyHotkeyCallbacks() 180
PtDestroyRawCallbacks() 181
PtDivider
    methods
      Extent 91
PtEventHandler() 75
PtExtentWidget() 91, 317
PtFindNextWidgetData() 182
PtFindResource() 183
PtFindWidgetData() 185
PtGauge 105
    methods
      Calc Opaque Rect 106
      Connection 105
      Defaults 105
      Destruction 105
      Draw 105
      Extent 105
      Got Focus 106
      Initialization 105
      Lost Focus 106
      Realization 105
      Unrealization 105
    widget actions 106
PtGenList 89, 109–117
    methods
      Calc Opaque Rect 92
      Connection 92
      Defaults 91
      Destruction 92
      Draw 92
      Extent 91
      Got Focus 92
      Initialization 91
      List Attributes method 94
      List Draw method 93
      List Inflate method 94
      List Key method 93
      List Mouse method 93
      List Select method 93
      Lost Focus 92
      Realization 92
      Unrealization 92
    widget actions 94
PtGenListAddItems() 110
PtGenListCreateTextBalloon() 115
PtGenListSetColumnBalloon() 115
PtGenListSetGflags() 110
PtGenListShow() 110
PtGenListUnlockItem() 110
PtGenListUnselect() 109
PtGenTree 96, 129–133
    methods
      Calc Opaque Rect 98
      Connection 97
      Defaults 97
      Destruction 98
      Draw 98
      Extent 97
      Got Focus 98
      Initialization 97
      List Draw 99, 129, 130
      List Key 99
      List Mouse 99
      Lost Focus 98
      Realization 97
      Tree Draw Item 99
      Tree Item State 99
      Unrealization 98
    widget actions 99
PtGenTreeAddFirst() 129
PtGenTreeCollapse() 99, 131
PtGenTreeExpand() 99, 131
PtGenTreeItem_t 129
PtGetResources() 30, 38
PtGetStruct() 187
PtGraphic 103
    methods
      Calc Opaque Rect 104
      Connection 103
      Defaults 103
      Destruction 104
      Draw 104
      Extent 103
      Got Focus 104
      Initialization 103
      Lost Focus 104
      Realization 103
      Unrealization 104
    widget actions 104
<table>
<thead>
<tr>
<th>PtGroup</th>
<th>methods</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Child Created</td>
<td>57</td>
</tr>
<tr>
<td>PtInvokeCallbackList()</td>
<td>79, 186, 189</td>
<td></td>
</tr>
<tr>
<td>PtInvokeCallbackType()</td>
<td>190</td>
<td></td>
</tr>
<tr>
<td>PtInvokeResizeCallbacks()</td>
<td>191</td>
<td></td>
</tr>
<tr>
<td>PtLabel</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td></td>
<td>methods</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Calc Opaque Rect</td>
<td>101</td>
</tr>
<tr>
<td></td>
<td>Connection</td>
<td>101</td>
</tr>
<tr>
<td></td>
<td>Defaults</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>Destruction</td>
<td>51</td>
</tr>
<tr>
<td></td>
<td>Draw</td>
<td>101</td>
</tr>
<tr>
<td></td>
<td>Extent</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>Got Focus</td>
<td>101</td>
</tr>
<tr>
<td></td>
<td>Initialization</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>Lost Focus</td>
<td>101</td>
</tr>
<tr>
<td></td>
<td>Realization</td>
<td>101</td>
</tr>
<tr>
<td></td>
<td>Unrealization</td>
<td>50, 101</td>
</tr>
<tr>
<td></td>
<td>widget actions</td>
<td>101</td>
</tr>
<tr>
<td>PtMenu</td>
<td>methods</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Connection</td>
<td>46</td>
</tr>
<tr>
<td>PtMenuBar</td>
<td>child-redirector function</td>
<td>59</td>
</tr>
<tr>
<td>PtModifyFlags()</td>
<td>77</td>
<td></td>
</tr>
<tr>
<td>PtMoveResizeWidget()</td>
<td>192</td>
<td></td>
</tr>
<tr>
<td>PtPrintSel</td>
<td>unexported subordinate children</td>
<td>44</td>
</tr>
<tr>
<td></td>
<td>RealizeWidget()</td>
<td>44–46, 317</td>
</tr>
<tr>
<td></td>
<td>RemoveWidgetData()</td>
<td>193</td>
</tr>
<tr>
<td></td>
<td>ResizeCanvas()</td>
<td>46</td>
</tr>
<tr>
<td></td>
<td>ResizePolicy()</td>
<td>195</td>
</tr>
<tr>
<td>PtResourceRec_t</td>
<td>28</td>
<td></td>
</tr>
<tr>
<td>PtSampCompound</td>
<td>redirecting children</td>
<td>67</td>
</tr>
<tr>
<td></td>
<td>redirecting resources</td>
<td>66</td>
</tr>
<tr>
<td>PtSetArg()</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>PtSetExtentFromArea()</td>
<td>196</td>
<td></td>
</tr>
<tr>
<td>PtSetResources()</td>
<td>29, 38, 73, 317</td>
<td></td>
</tr>
<tr>
<td>PtSetStruct()</td>
<td>77, 197, 199</td>
<td></td>
</tr>
<tr>
<td>PtSetValue()</td>
<td>197, 199</td>
<td></td>
</tr>
<tr>
<td>PtSuperClassCalcOpaque()</td>
<td>101, 200</td>
<td></td>
</tr>
<tr>
<td>PtSuperClassChildCreated()</td>
<td>201</td>
<td></td>
</tr>
<tr>
<td>PtSuperClassChildDestroyed()</td>
<td>202</td>
<td></td>
</tr>
<tr>
<td>PtSuperClassChildGettingFocus()</td>
<td>203</td>
<td></td>
</tr>
<tr>
<td>PtSuperClassChildGettingResources()</td>
<td>204</td>
<td></td>
</tr>
<tr>
<td>PtSuperClassChildLosingFocus()</td>
<td>205</td>
<td></td>
</tr>
<tr>
<td>PtSuperClassChildMovedResized()</td>
<td>206</td>
<td></td>
</tr>
<tr>
<td>PtSuperClassChildRealized()</td>
<td>208</td>
<td></td>
</tr>
<tr>
<td>PtSuperClassChildSettingResized()</td>
<td>209</td>
<td></td>
</tr>
<tr>
<td>PtSuperClassChildUnrealized()</td>
<td>210</td>
<td></td>
</tr>
<tr>
<td>PtSuperClassConnect()</td>
<td>211</td>
<td></td>
</tr>
<tr>
<td>PtSuperClassConnectFrom()</td>
<td>211</td>
<td></td>
</tr>
<tr>
<td>PtSuperClassDraw()</td>
<td>20, 101, 212</td>
<td></td>
</tr>
<tr>
<td>PtSuperClassExtent()</td>
<td>78, 83, 91, 214</td>
<td></td>
</tr>
<tr>
<td>PtSuperClassGenListDraw()</td>
<td>118</td>
<td></td>
</tr>
<tr>
<td>PtSuperClassGenListInflate()</td>
<td>119</td>
<td></td>
</tr>
<tr>
<td>PtSuperClassGenListKey()</td>
<td>121</td>
<td></td>
</tr>
<tr>
<td>PtSuperClassGenListMouse()</td>
<td>123</td>
<td></td>
</tr>
<tr>
<td>PtSuperClassGenListSelect()</td>
<td>125</td>
<td></td>
</tr>
<tr>
<td>PtSuperClassGenTreeDrawItem()</td>
<td>134</td>
<td></td>
</tr>
<tr>
<td>PtSuperClassGenTreeItemState()</td>
<td>135</td>
<td></td>
</tr>
<tr>
<td>PtSuperClassGetResources()</td>
<td>216</td>
<td></td>
</tr>
<tr>
<td>PtSuperClassGotFocus()</td>
<td>92, 217</td>
<td></td>
</tr>
<tr>
<td>PtSuperClassInit()</td>
<td>218</td>
<td></td>
</tr>
<tr>
<td>PtSuperClassInitFrom()</td>
<td>218</td>
<td></td>
</tr>
<tr>
<td>PtSuperClassLostFocus()</td>
<td>52, 92, 219</td>
<td></td>
</tr>
<tr>
<td>PtSuperClassRawEvent()</td>
<td>80, 94, 220</td>
<td></td>
</tr>
<tr>
<td>PtSuperClassRawEventFrom()</td>
<td>220</td>
<td></td>
</tr>
<tr>
<td>PtSuperClassRealized()</td>
<td>221</td>
<td></td>
</tr>
<tr>
<td>PtSuperClassSetResources()</td>
<td>81, 197, 199, 222</td>
<td></td>
</tr>
<tr>
<td>PtTimer</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PtWidget</td>
<td>45, 73</td>
<td></td>
</tr>
<tr>
<td></td>
<td>extending</td>
<td>41</td>
</tr>
<tr>
<td></td>
<td>methods</td>
<td>42</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
R

raw callbacks 37, 39, 54, 55, 80, 87, 94, 147, 220
  freeing 181
PtBasic 79
PtCompound 89
PtContainer 85
PtGauge 106
PtGenTree 99
PtGraphic 104
PtLabel 101
PtWidget 74
Realization method 16, 37, 39, 48, 221
  inheriting 17
PtBasic 78
PtComboBox 69
PtCompound 69, 88
PtContainer 62, 83
PtGauge 105
PtGenList 92
PtGenTree 97
PtGraphic 103
PtLabel 101
PtWidget 74
ShadowedBox 18
redirection function
  resource 66
redirector function
  child 9, 22, 59, 66
    PtMenuBar 59
regions
  windows
    changing attributes 157
    closing 159
    opening 160
regions, determining if required 170, 175
resize callbacks 83
  invoking 191
resource
  editor
    open function 270
resource editors
  base structure 275
  bringing to front 296, 300
  comparing values 258
  getting changes 292
  setting data 294
  state
    setting 298
resource-redirector function 66
resources
  blocking 23, 62, 63, 66, 88
  children 9
  exported subordinates 10
  causing damage 20
Child Getting Resource method 59
Child Setting Resource method 58
class resource table 39
default values in PhAB 149
defining 4, 15, 27
describing in PhAB 141, 143
defining in PhAB 139, 140
freeing 51, 74
freeing memory 31
Get Resources method 51
getting 66
  inheriting 4, 147
manifests 28
numbers 28
Pt_NO_INHERITED_RESOURCES 39
PtBasic 80
PtContainer 85
PtCreateWidget() 15
PtFindResource() 183
PtGauge 106
PtGenList 94
PtGenTree 99
PtGetStruct() 187
PtLabel 101
PtSetStruct() 197
PtSetValue() 199
PtSuperClassChildGettingResources() 204
PtSuperClassChildSettingResources() 209
PtSuperClassGetResources() 216
PtSuperClassSetResources() 222
records 28
Set Resources method 51, 69
types 30
RESPLUGIN_ACTION_APPLY 259
RESPLUGIN_ACTION_CLOSE 259
RESPLUGIN_ACTION_DEFAULT  259
RESPLUGIN_CHANGES  254
RESPLUGIN_FIXED_SIZE  259
RESPLUGIN_NO_CHANGES  254
RESPLUGIN_STATE_BLOCKED  298
RESPLUGIN_STATE_DISABLED  298
RESPLUGIN_STATE_MATCH_DEFAULT  298
RESPLUGIN_STATE_MATCH_DEFAULT_MASTER  298
RESPLUGIN_STATE_MATCH_MASTER  298
RESPLUGIN_STATE_NO_MATCH  298
RESPLUGIN_STATE_UNBLOKED  298

S

Set Resources method  39, 51
  inheriting  17
  PtCompound  69
    PtFindResource()  183
setting
  resource editor data  294
setting states
  resource editors  298
ShadowedBox
  class resource table  41
  class structure  6
  class-creation function  14, 176
  defined  5–8
  life cycle  13
  methods
    chaining  16
    Connection  18
    Defaults  8, 15, 42
    Draw  18, 42
    Extent  18
    inheriting  17
    Initialization  18
    Realization  18
    Unrealization  21
resource declarations  33
shared object, creating  140
source file  6
  sample code  6–8
structures
  class  6
instance  6
subordinate widgets
  blocking resources  10, 23
  exported  10, 22, 62, 87
  blocking resources  66
  creating  67
  distinguishing resources  66
  identifying  23
  Pt_ARG_DATA  67
  Pt_PROCREDATED  67
  Pt_SET_NUM_SUBORDINATES  67, 88
  Pt_SET_SUBORDINATES  67, 88
  PtValidParent()  67
  realizing  69
  setting resources  317
  unexported  44

T

tips for building widgets  317
  to front
    resource editors  296, 300
Tree Draw Item method
  PtGenTree  99
    PtSuperClassGenTreeDrawItem()  134
tree widgets  129, 130
Tree Item State method
  PtGenTree  99
    PtSuperClassGenTreeItemState()  135
tree widgets  129–133, See also PtGenTree
  methods
    List Draw  129
    Tree Draw Item  129, 130
    Tree Item State  131
typographical conventions  xvi

U

Unrealization method  21, 37, 39, 50
  chaining  17
  PtBasic  79
  PtCompound  89
### Index

<table>
<thead>
<tr>
<th>PtContainer</th>
<th>84</th>
</tr>
</thead>
<tbody>
<tr>
<td>PtGauge</td>
<td>105</td>
</tr>
<tr>
<td>PtGenList</td>
<td>92</td>
</tr>
<tr>
<td>PtGenTree</td>
<td>98</td>
</tr>
<tr>
<td>PtGraphic</td>
<td>104</td>
</tr>
<tr>
<td>PtLabel</td>
<td>50, 101</td>
</tr>
<tr>
<td>PtWidget</td>
<td>50, 74</td>
</tr>
<tr>
<td>ShadowedBox</td>
<td>21</td>
</tr>
</tbody>
</table>

**W**

widgets

- actions 37, 54
- adding to PhAB 139
- anchoring 9, 13
- basic 9
- canvas
  - resizing 194
- canvas, calculating 171
- categories 8
- children
  - bounding box 171
- classes
  - defining 36–42
  - description 40
  - methods 42–54
- compound 10
- container 9
- damaging 178
- data chain 162, 182, 185, 193
- description table 141
- designing 4
- extent, synchronizing 192
- handling events 37, 54
- instantiating 14
- life cycle 13
- main components 5
- managing children 9
- reasons to customize 3
- redirecting 10
- region, determining if required 170, 175
- repairing damage 20
- resizing, attempting 167
- resizing policy 195
- resources blocking 9, 10, 23, 62, 63, 66, 88
- causing damage 20
- Child Getting Resource method 59
- Child Setting Resource method 58
- class resource table 39
- default values in PhAB 149
- defining 4, 15, 27
- describing in PhAB 141, 143
- editing in PhAB 139, 140
- freeing 51, 74
- freeing memory 31
- Get Resources method 51, 69
- getting 66
- inheriting 4, 147
- manifests 28
- numbers 28
- Pt_NO_INHERITED_RESOURCES 39
- PtBasic 80
- PtContainer 85
- PtCreateWidget() 15
- PtFindResource() 183
- PtGauge 106
- PtGenList 94
- PtGenTree 99
- PtGetStruct() 187
- PtGraphic 104
- PtLabel 101
- PtSetStruct() 197
- PtSetValue() 199
- PtSuperClassChildGettingResources() 204
- PtSuperClassChildSettingResources() 209
- PtSuperClassGetResources() 216
- PtSuperClassSetResources() 222
- records 28
- Set Resources method 51, 69
- subclassing 3
- PtBasic 3
- PtCompound 10
- PtContainer 9
- PtGenList 89
- PtWidget 9
- superclass types 8
- tips for building 317
- visibility, change in 224
widget in front of window 226
area in PhAB plugins 266
windows regions changing attributes 157
closing 159
opening 160