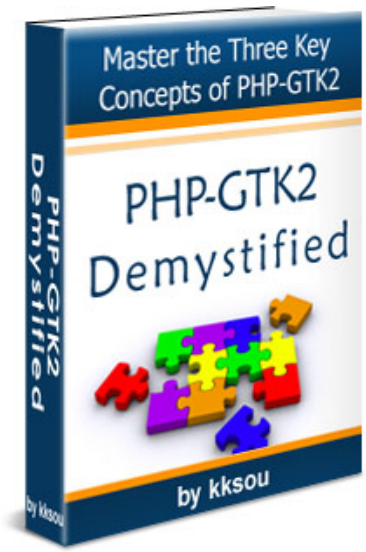
PHP-GTK2 Demystified

Understand the three Key Concepts of PHP-GTK2



Another original product from

www.kksou.com

Limits of Liability / Disclaimer of Warranty:

The authors and publisher of this book and the accompanying materials have used their best efforts in preparing this program. The authors and publisher make no representation or warranties with respect to the accuracy, applicability, fitness, or completeness of the contents of this program. They disclaim any warranties (expressed or implied), merchantability, or fitness for any particular purpose. The authors and publisher shall in no event be held liable for any loss or other damages, including but not limited to special, incidental, consequential, or other damages. As always, the advice of a competent legal, tax, accounting or other professional should be sought.

Table of Contents

Preface

I wrote this book in the hope that it will save you some time picking up PHP-GTK2.

I've been developing applications using PHP for years. When I first encountered PHP-GTK2, I was thrilled at the thought of being able to develop cross-platform desktop applications using my knowledge of PHP.

Getting "hello world" to run is easy enough. However, the moment I started to develop serious applications using PHP-GTK2, I was stuck! I find that I couldn't even get simple stuff working - such as positioning and setting the size of the widgets. Due to the lack of documentation on PHP-GTK, I had to spend countless hours of research on the net, asking friends, and going through numerous trials and errors.

This is why I've written this book, so that you do not need to "grope in the dark" like what I've been through in picking up PHP-GTK2.

What This Book Covers

This book focuses on the three most important concepts in PHP-GTK2:

- Size and positioning of widgets
- Callback functions, and
- Object-oriented framework.

If you've worked with PHP-GTK2, this book will help you gain an in-depth understanding of these three key concepts. The more you use PHP-GTK, the more you will find that all PHP-GTK programming boils down to a mastery in these three areas.

If you are new to PHP-GTK2, this book will be an excellent tutorial for you. You will start to program in PHP-GTK2 from the very first lesson, as I slowly introduce the various key concepts in PHP-GTK2 one by one. By the end of the book, you will be equipped with a solid foundation that enables you start building serious applications using PHP-GTK2.

Feedback

I have tested and verified the information in this book to the best of my knowledge. However, no one is perfect, and mistakes do occur. If you find any errors in the book, such as typos, inaccuracies, bugs, misleading or confusing statements, please help to improve future editions by sending me your feedback to feedback@kksou.com.

Chapter 1 Getting Started

1.1 Hello World!

Objective

Let's get started learning php-gtk2 with this simple yet complete "hello world!" script.

We will also understand the four key lines that constitute a standard php-gtk2 script. You will see these four lines in almost all the sample codes in this book.

Overview

- First create a new GtkWindow.
- Then create a new GtkLabel.
- Stuff the label inside the window with GtkContainer::add().

Sample Output

Sample Code

```
1  <?php
2  $window = new GtkWindow(); // note 1
3  $window->connect_simple('destroy',array('Gtk','main_quit')); // note 2
4
5  // add your widgets here
6  $label = new GtkLabel("hello world!");
7  $window->add($label);
8
9  $window->show_all(); // note 3
10 Gtk::main(); // note 4
11 ?>
```

Listing 1.1.php

Explanation

For almost all php-gtk2 scripts, you will find the following four key lines (highlighted in blue):

1. Creates a new window.
2. Ensures a clean exit when you close the window. This basically says call `Gtk::main_quit()` when the user close the window.
3. Displays all widgets that you have created, in this case, the window and the label.

4. Let GTK take over and start waiting for events (e.g. mouse or keyboard input).

Note

- These four lines form the simplest yet complete php-gtk2 script. For this “hello world” example, we simply stuff two more lines of code between these 4 lines:

```
$label = new GtkLabel("hello world!");
$window->add($label);
```

  The first to create a new label, and the second to stuff the label inside the window.

- You can also combine these two lines into one:

```
$window->add(new GtkLabel("hello world!"));
```

1.2 Display a button

Objective

Let's use another widget in this example - a button.

Overview

In the previous example, we use GtkLabel to display "Hello World". In this example, we use GtkButton to display a button. The GtkLabel and GtkButton are called widgets in php-gtk. There are many other **widgets** in such as GtkRadioButton, GtkMenu, GtkTable, including GtkWindow.

- You will see the same 4 key lines (as explained in the previous example).
- Now instead of creating a GtkLabel, we create a GtkButton.
- Then stuff the button inside the window using GtkContainer::add() as before.

Sample Output

Sample Code

```
1  <?php
2  $window = new GtkWindow();
3  $window->connect_simple('destroy',array('Gtk','main_quit'));
4
5  $button = new GtkButton("click me!"); // note 1
6  $window->add($button); // note 2
```



```
7
8  $window->show_all();
9  Gtk::main();
10 ?>
```

Listing 1.2.php

Explanation

1. Creates a button.
2. Add the button to the window.

Note

Think of widgets as different types of Lego building blocks. Some are plain rectangular blocks. Some are round blocks. Some have buttons. By creatively combining these different building blocks, we can build almost anything we have in mind.

Same for php-gtk2. Writing php-gtk2 scripts is all about knowing what are the widgets available, and their corresponding properties, methods and signals.

1.3 Responding to button click

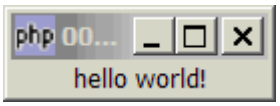
Objective

We've displayed a button in the previous example. Now we will respond to button clicks.

Overview

- When a user presses the button, a **signal** clicked is generated. Think of signal as something php-gtk generates to inform you that something has occurred, in this example, the user has clicked the button.
- By connecting the signal to a callback function, every time there is a 'clicked' signal, php-gtk will automatically call the callback function that you have connected.

Sample Output



Sample Code

```
1  <?php
2  $window = new GtkWindow();
3  $window->connect_simple('destroy',array('Gtk','main_quit'));
4
```



```
5  $button = new GtkButton("click me!");
6  $button->connect('clicked', 'on_click'); // note 1
7
8  $window->add($button);
9  $window->show_all();
10 Gtk::main();
11
12 function on_click($button) { // note 2
13    echo "button clicked!\n";
14 }
15 ?>
```

Listing 1.3.php

Explanation

1. Connects the signal 'clicked' to the callback function `on_click()`.
2. This is the callback function that is called when the user clicks on the button. Here we simply echo the words "button clicked!" to the command window.

Note

- Different signals are generated by a widget for different types of events. Each signal has a unique name assigned to it. E.g. for GtkButton:
    - the signal clicked is emitted when the user clicks the button.
    - the signal pressed is emitted when the button is being pressed.
    - the signal released is emitted when the button is being released.
- You will sometimes see people referring callback functions as **signal handlers**, since these callback functions are used to handle signals. signal handlers

1.4 Adding two or more widgets

Objective

Up until now, we have only added one widget to GtkWindow - either a label or a button. Suppose we want to add both the label and the button.

Overview

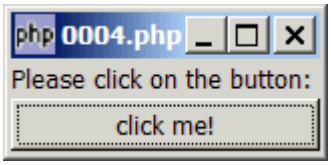
The GtkWindow widget is called a **bin** in php-gtk. A bin can contain one and only one child widget. To stuff more than one widgets in a GtkWindow, we need **containers** — widgets that can hold two or more widgets.

In this example, we will use a GtkVBox. As the name suggests, this is a vertical container that can be used to “pack” widgets vertically.

- Create a GtkVBox.

- Use GtkBox::pack_start() to add widgets into vbox. The widgets will be packed into the vbox starting from the top.

Sample Output



Sample Code

```
1  <?php
2  $window = new GtkWindow();
3  $window->connect_simple('destroy',array('Gtk','main_quit'));
4  $vbox = new GtkVBox(); // note 1
5  $window->add($vbox); // note 2
6
7  $vbox->pack_start(new GtkLabel('Please click on the button: ')); // note 3
8
9  $button = new GtkButton("click me!");
10 $button->connect('clicked', 'on_click');
11 $vbox->pack_start($button); // note 4
12
13 $window->show_all();
14 Gtk::main();
15
16 function on_click($button) {
17    echo "button clicked!\n";
18 }
19 ?>
```

Listing 1.4.php

Explanation

1. Create the vbox.
2. Add the vbox to the GtkWindow.
3. First stuff the label into the vbox.
4. Followed by the button.

Note

- If you wish to pack the widget from the bottom stacking up, use the method GtkBox::pack_end().
- Commonly used bin: GtkWindow, GtkFrame, GtkEventBox, GtkAlignment, GtkScrolledwindow.

- Commonly used containers: GtkVBox, GtkHBox, GtkTable, GtkTreeview, GtkMenu.

1.5 Resize application window

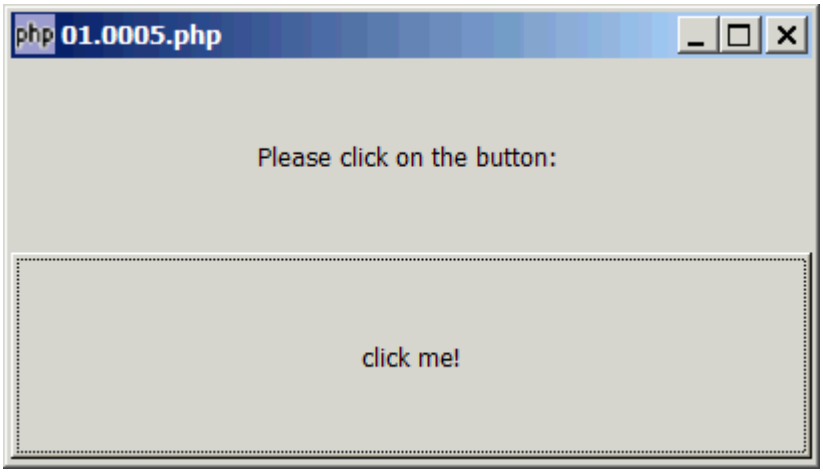
Objective

We will resize the application window to 400 x 200 in this example.

Overview

Resize widgets (including GtkWindow) is easy with a call to the method GtkWidget::set_size_request().

Sample Output



Sample Code

```
1  <?php
2  $window = new GtkWindow();
3  $window->connect_simple('destroy',array('Gtk','main_quit'));
4  $window->set_size_request(400, 200); // note 1
5  $vbox = new GtkVBox();
6  $window->add($vbox);
7
8  $label = new GtkLabel('Please click on the button: ');
9  //$label->set_size_request(100,50); // note 2
10 $vbox->pack_start($label);
11
12 $button = new GtkButton("click me!");
13 //$button->set_size_request(100,50); // note 2
14 $button->connect('clicked', 'on_click');
15 $vbox->pack_start($button);
16
17 $window->show_all();
```



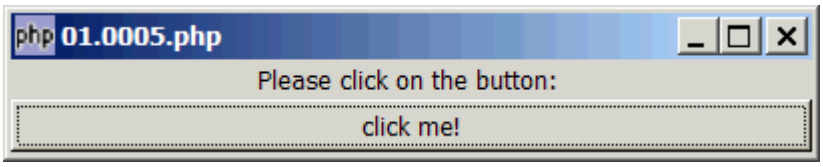
```
18 Gtk::main();
19
20 function on_click($button) {
21     echo "button clicked!\n";
22 }
23 ?>
```

Listing 1.5.php

Explanation

1. Resize the application window to 400x200.

   Note that if you specify only the width, e.g. `$window->set_size_request(400,-1)` you will get the following.

   If you specify only the height, e.g. `$window->set_size_request(-1,200), you will get:`

2. Please see notes below.

Note

The method GtkWidget::set_size_request() is supposed to work with all GtkWidgets, including GtkWindow, GtkLabel and GtkButtons. However, easy as it might seems, this is one area that frustrates a lot of people new to php-gtk.

- Sometimes you "requested" the size for a widget, but the size remains unchanged. For example, try uncomment the two lines with "note 2" above. You will find that the label and the button remain as big!
- Sometimes you changed the size of a container, and the widgets inside the containers automatically get resized too, messing up the layout and positioning.

> For example, the button in the previous example looks ok. But as soon as you resize the application window, the button becomes too huge.

Of course, there are always two sides to a coin. Please take a look at this example. A user can choose the calculator button size he or she likes by resizing the application window. Php-gtk2 automatically rearranges and resizes the widgets to fit the new window - without the need of you to do any programming!

So, how do we set the size of the label and button in php-gtk?

This leads us to the first fundamental building block of php-gtk: **Size and Positioning**.

Chapter 2 Size & Positioning

In developing any desktop applications, the first task is usually laying out the widgets.

It is easy to get a couple of widgets up and running in php-gtk, as we have seen in the examples in Chapter 1. However, for someone new to php-gtk, he or she will soon find out that some supposedly simple tasks – such as setting the size and position of widgets – are not that simple after all.

2.1 Understand the *Expand* parameter

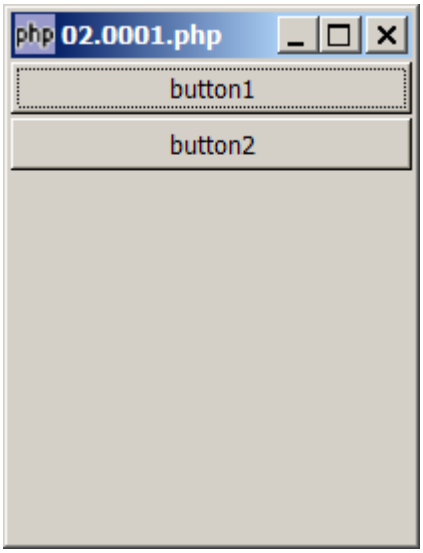
Objective

Let's first understand the *expand* parameter.

Overview

- First create two GtkButton.
- This time when packing the buttons into the vbox, use pack_start($button, *false*).

Sample Output



Sample Code

```
1  <?php
2  $window = new GtkWindow();
3  $window->connect_simple('destroy',array('Gtk','main_quit'));
4  $window->set_size_request(400, 200);
5  $vbox = new GtkVBox();
6  $window->add($vbox);
7
8  $vbox->pack_start(new GtkButton('button1'), false); // note 1
```



```
9  $vbox->pack_start(new GtkButton('button2'), false); // note 2
10
11 $window->show_all();
12 Gtk::main();
13 ?>
```

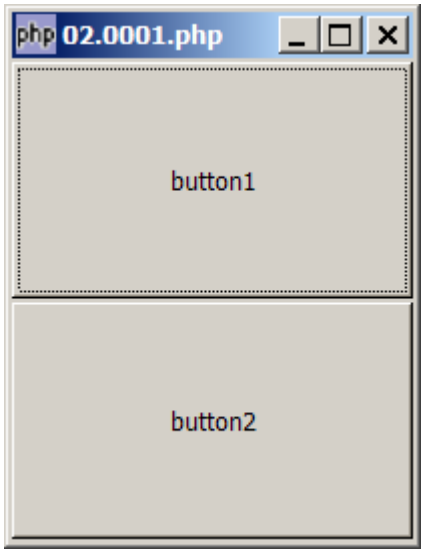
Listing 2.1.php

Explanation

1. Pack button1 and button2 into vbox. Note that we set the *expand* parameter to `false`.

Compare the above with that of setting the `expand` parameter to `true` as shown in the figure on the right.

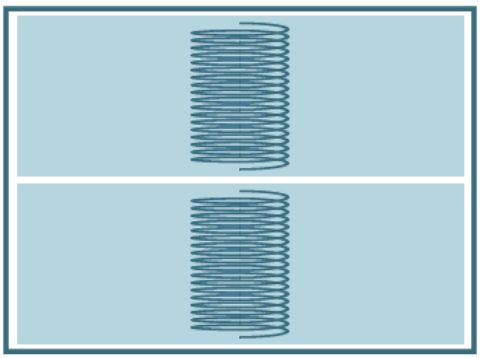
See the difference?



The "Spring Box" Model

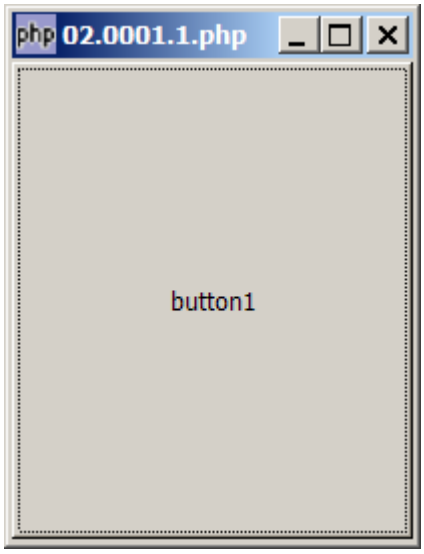
The best way to understand the expand parameter is to think of a button with `expand` set to `true` as a button with a spring inside, as shown in the diagram on the right.

Because of the spring, a button with `expand=true` will fight for any space available to it.



One Expandable Button

So for the code below, you will find that button1 will grab the entire window space because of the "expandable spring" inside it.

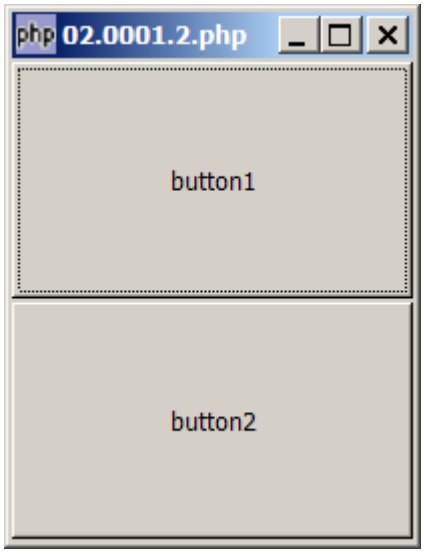



```
1  <?php
2  $window = new GtkWindow();
3  $window-
   >connect_simple('destroy',array('Gtk','main_quit'));
4  $window->set_size_request(200, 100);
5  $vbox = new GtkVBox();
6  $window->add($vbox);
7
8  $vbox->pack_start(new GtkButton('button1'), true);
9
10 $window->show_all();
11 Gtk::main();
12 ?>
```

Listing 2.1.1.php

Two Expandable Buttons

If we have two buttons, both set to `expand=true`, you will find that button1 and button2 will both fight for the available space. As the two springs are of equal strength, the result is that each will grab half the space.




```
1  <?php
2  $window = new GtkWindow();
3  $window-
   >connect_simple('destroy',array('Gtk','main_quit'));
4  $window->set_size_request(200, 100);
5  $vbox = new GtkVBox();
6  $window->add($vbox);
7
8  $vbox->pack_start(new GtkButton('button1'), true);
9  $vbox->pack_start(new GtkButton('button2'), true);
10
11 $window->show_all();
12 Gtk::main();
13 ?>
```

Listing 2.1.2.php

Three Expandable Buttons

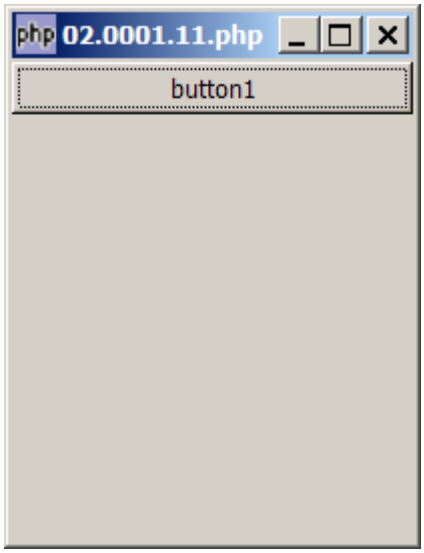
As you have probably guessed, for three buttons all set to `expand=true`, the result is as shown on the right:


```
1  <?php
2  $window = new GtkWindow();
3  $window-
   >connect_simple('destroy',array('Gtk','main_quit'));
4  $window->set_size_request(200, 100);
5  $vbox = new GtkVBox();
6  $window->add($vbox);
7
8  $vbox->pack_start(new GtkButton('button1'), true);
9  $vbox->pack_start(new GtkButton('button2'), true);
10 $vbox->pack_start(new GtkButton('button3'), true);
11
12 $window->show_all();
13 Gtk::main();
14 ?>
```

Listing 2.1.3.php

One Button – no spring

When you set `expand=false`, think of this as a button without any spring inside. Since there are no expandable spring, the widget will take on its default size:

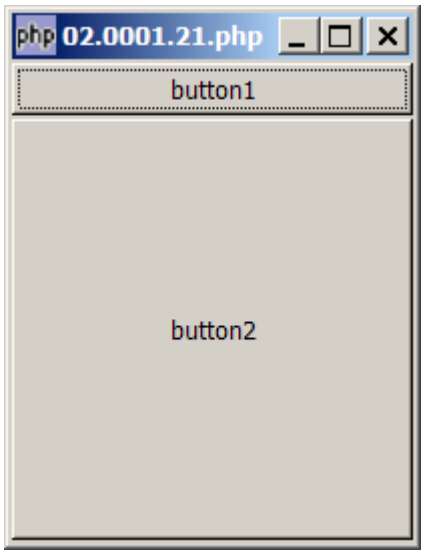



```
1  <?php
2  $window = new GtkWindow();
3  $window-
   >connect_simple('destroy',array('Gtk','main_quit'));
4  $window->set_size_request(200, 100);
5  $vbox = new GtkVBox();
6  $window->add($vbox);
7
8  $vbox->pack_start(new GtkButton('button1'), false);
9
10 $window->show_all();
11 Gtk::main();
12 ?>
```

Listing 2.1.4.php

Two Buttons – first no spring, second with spring

Suppose now you have two buttons, the first one with `expand=false` and the second with `expand=true`. Because of the spring in the second button, it will expand – pushing the first button all the way up, as shown in the figure on the right:

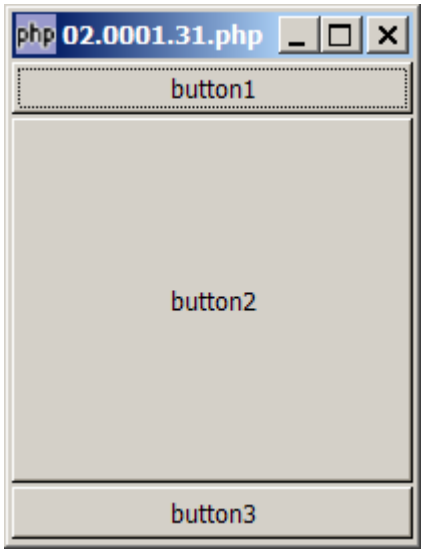



```
1  <?php
2  $window = new GtkWindow();
3  $window-
   >connect_simple('destroy',array('Gtk','main_quit'));
4  $window->set_size_request(200, 100);
5  $vbox = new GtkVBox();
6  $window->add($vbox);
7
8  $vbox->pack_start(new GtkButton('button1'), false);
9  $vbox->pack_start(new GtkButton('button2'), true);
10
11 $window->show_all();
12 Gtk::main();
13 ?>
```

Listing 2.1.5.php

Three Buttons – on, off, on

Suppose now you have three buttons, the first and third with `expand=false` and the second with `expand=true`. Because of the spring in the second button, it will expand – pushing the first button all the way up and the third button all the way down:




```
1  <?php
2  $window = new GtkWindow();
3  $window-
   >connect_simple('destroy',array('Gtk','main_quit'));
4  $window->set_size_request(200, 100);
5  $vbox = new GtkVBox();
6  $window->add($vbox);
7
8  $vbox->pack_start(new GtkButton('button1'), false);
9
10 $window->show_all();
11 Gtk::main();
12 ?>
```

Listing 2.1.6.php

Same for GtkLabel

The "spring box" model applies to GtkLabel too. Because a label has no borders like that of a button, you only see the text. But the effect is the same.


```
1  <?php
2  $window = new GtkWindow();
3  $window-
   >connect_simple('destroy',array('Gtk','main_quit'));
4  $window->set_size_request(200, 100);
5  $vbox = new GtkVBox();
6  $window->add($vbox);
7
8  $vbox->pack_start(new GtkLabel('label1'), false);
9  $vbox->pack_start(new GtkLabel('label2'), true);
10 $vbox->pack_start(new GtkLabel('label3'), false);
11
12 $window->show_all();
13 Gtk::main();
14 ?>
```

Listing 2.1.7.php

Important Note

Understanding the **expand** parameter is key to understanding layout and positioning in php-gtk. So it is important that you understand all the examples in this article before moving on.

2.2 Understand the *Fill* parameter

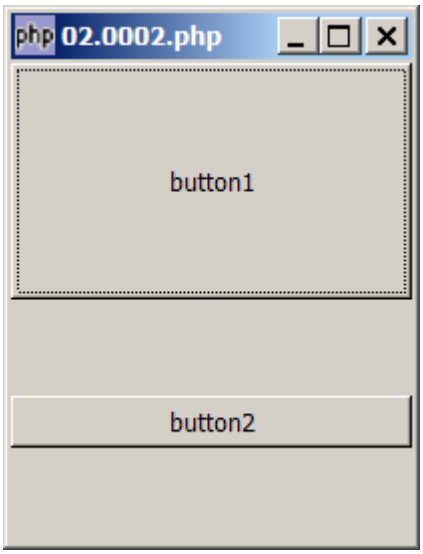
Objective

Now that we've understood the *expand* parameter. Let's move on to the *fill* parameter.

Overview

- First create two GtkButton.
- For the first button, set `expand=true` and `fill=false`.
- For the second button, set `expand=true` and `fill=true`.

Sample Output



Sample Code

```
1  <?php
2  $window = new GtkWindow();
3  $window->connect_simple('destroy',array('Gtk','main_quit'));
4  $window->set_size_request(200, 100);
5  $vbox = new GtkVBox();
6  $window->add($vbox);
7
8  $vbox->pack_start(new GtkButton('button1'), true, true); // note 1
9  $vbox->pack_start(new GtkButton('button2'), true, false); // note 2
10
11 $window->show_all();
12 Gtk::main();
13 ?>
```

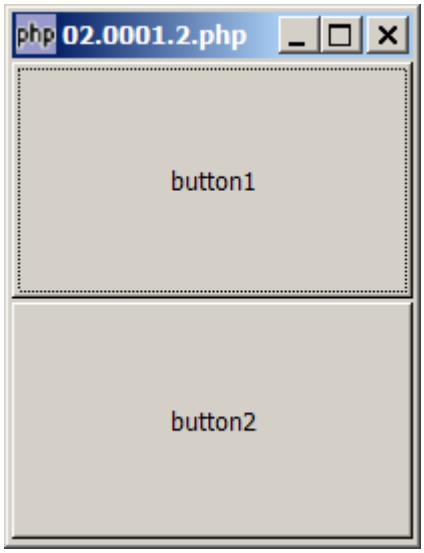
Listing 2.2.php

Explanation

1. For both buttons, we set `expand=true`. So as explained in the previous article, both will get equal share of the available spaces. For button 1, we set `fill=true`, meaning the button will fill up the entire space.
2. For button 2, it will also get half the share of the space (because we set expand=true). However, because we have set `fill=false`, it will NOT fill up the entire space, i.e. it will retain its default size.

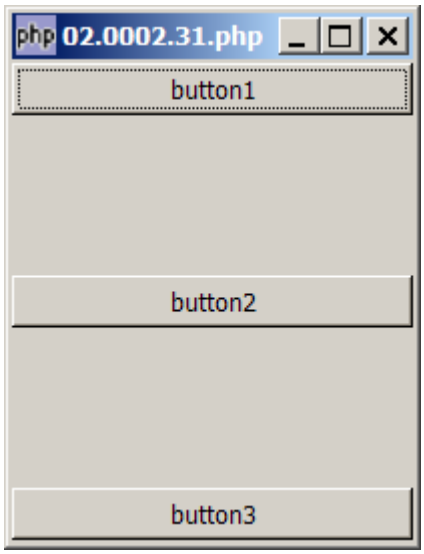
Compare the above with that of setting `expand=true` and `fill=true` as shown in the figure on the right.

See the difference?



Three Buttons – fill = on, off, on

In the example below, only button2 is expandable, but with `fill=false`. So we have button2 getting all the spaces, but retaining its original size:



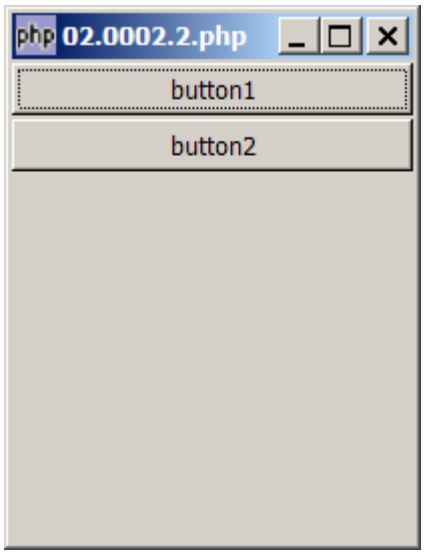

```
1  <?php
2  $window = new GtkWindow();
3  $window-
   >connect_simple('destroy',array('Gtk','main_quit'));
4  $window->set_size_request(200, 100);
5  $vbox = new GtkVBox();
6  $window->add($vbox);
7
8  $vbox->pack_start(new GtkButton('button1'), false, true);
9  $vbox->pack_start(new GtkButton('button2'), true, false);
10 $vbox->pack_start(new GtkButton('button3'), false, true);
11
12 $window->show_all();
13 Gtk::main();
14 ?>
```

Listing 2.2.1.php

Two Buttons – both expand=off

Note that the parameter `fill` has meaning only when `expand=true`. When `expand=false` and you set `fill=true`, you won't be able to see any effect as shown below:

```
1  <?php
2  $window = new GtkWindow();
3  $window-
   >connect_simple('destroy',array('Gtk','main_quit'));
4  $window->set_size_request(200, 100);
5  $vbox = new GtkVBox();
6  $window->add($vbox);
7
8  $vbox->pack_start(new GtkButton('button1'), false, true);
9  $vbox->pack_start(new GtkButton('button2'), false, true);
10
11 $window->show_all();
12 Gtk::main();
13 ?>
```



You will get the same result whether you set `fill=true` or fill=false when **`expand=false`**.

Listing 2.2.2.php

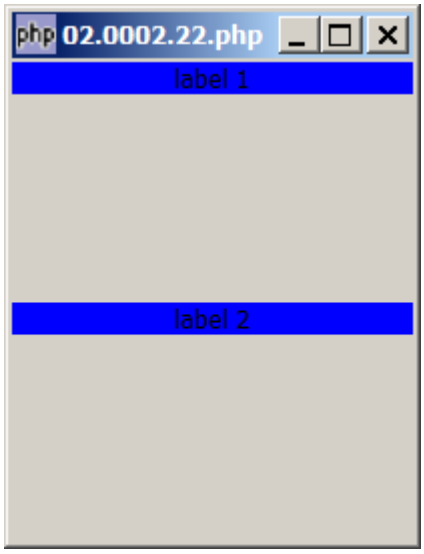
Two Labels – appears to have no effect for Fill parameter

Setting the `fill` parameter to `true` or `false` does not seem to have any effect on GtkLabel as shown below:

```
1  <?php
2  $window = new GtkWindow();
3  $window-
   >connect_simple('destroy',array('Gtk','main_quit'));
4  $window->set_size_request(200, 100);
5  $vbox = new GtkVBox();
6  $window->add($vbox);
7
8  $vbox->pack_start(new GtkLabel('label 1'), false, true);
9  $vbox->pack_start(new GtkLabel('label 2'), true, true);
10
11 $window->show_all();
12 Gtk::main();
13 ?>
```

Listing 2.2.3.php

Two Labels – see the *Fill* parameter in action



In the example above, the fill parameter does not seem to have any effect on GtkLabel. This is because a GtkLabel does not have a border like that of GtkButton. Hence it appears to have no effect. If we add a background color to the GtkLabel, we can see that it is working just as expected like the GtkButton.

```
1  <?php
2  $window = new GtkWindow();
3  $window-
   >connect_simple('destroy',array('Gtk','main_quit'));
4  $window->set_size_request(200, 100);
5  $vbox = new GtkVBox();
6  $window->add($vbox);
7
8  $eventbox1 = new GtkEventBox();
9  $eventbox1->add(new GtkLabel('label 1'));
10 $eventbox1->modify_bg(Gtk::STATE_NORMAL,
   GdkColor::parse("#0000ff"));
11 $vbox->pack_start($eventbox1, false, true);
12
13 $eventbox2 = new GtkEventBox();
14 $eventbox2->add(new GtkLabel('label 2'));
15 $eventbox2->modify_bg(Gtk::STATE_NORMAL,
   GdkColor::parse("#0000ff"));
16 $vbox->pack_start($eventbox2, true, false);
17
18 $window->show_all();
19 Gtk::main();
20 ?>
```

Listing 2.2.4.php

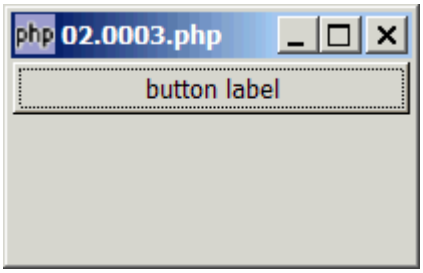
Important Note

Once you have understood the `expand` and the `fill` parameters, you will find that we have now absolute control over the size, layout and positioning of widgets in php-gtk2!

2.3 Display the button in default size

Objective

By right, the default size of a GtkButton corresponds to the size of its label. However, as you have seen in the previous few examples, it is always displayed extending across the entire width of the screen as shown below:

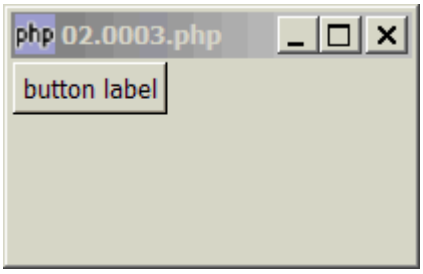


In this example, we will see how to display a button in its default size.

Overview

- First create a GtkButton.
- Then create a GtkHBox.
- Pack the button inside the hbox, set `expand=false`.
- Then pack the hbox inside the vbox, also set `expand=false`.

Sample Output



Sample Code

```
1  <?php
2  $window = new GtkWindow();
3  $window->connect_simple('destroy',array('Gtk','main_quit'));
4  $window->set_size_request(200, 100);
5  $vbox = new GtkVBox();
6  $window->add($vbox);
7
8  $button = new GtkButton('button label');
9  $hbox = new GtkHBox();
10 $hbox->pack_start($button, false); // note 1
11 $vbox->pack_start($hbox, false); // note 2
```



```
12
13 $window->show_all();
14 Gtk::main();
15 ?>
```

Listing 2.3.php

Explanation

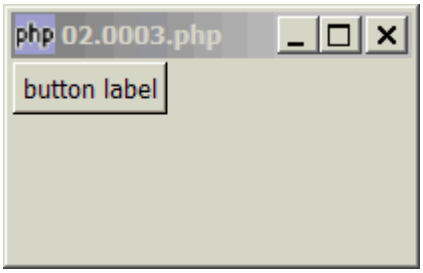
1. `expand=false` will ensure that the button does not expand horizontally in the hbox.
2. `expand=false` will ensure that the button does not expand vertically in the vbox.

As a result, the button is displayed in its default size – at the top, left corner of the window.

2.4 Right align the button

Objective

In the previous example, you have displayed the button in its default size as shown below:

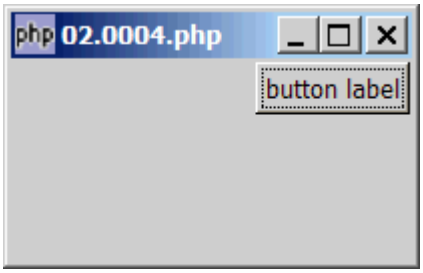


What if we want the button to be right aligned?

Overview

- Create a GtkButton.
- Create a GtkHBox.
- First pack an expandable box into the hbox. Remember to set `expand=true`.
- Then pack the button inside the hbox, set `expand=false`.
- Finally pack the hbox inside the vbox, also set `expand=false`.

Sample Output



Sample Code

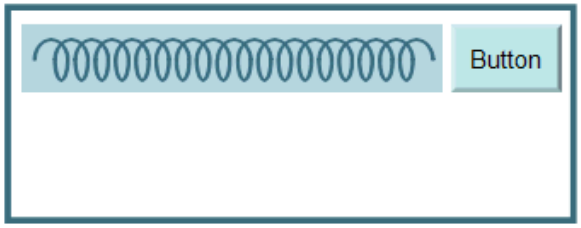
```
1  <?php
2  $window = new GtkWindow();
3  $window->connect_simple('destroy',array('Gtk','main_quit'));
4  $window->set_size_request(200, 100);
5  $vbox = new GtkVBox();
6  $window->add($vbox);
7
8  $button = new GtkButton('button label');
9  $hbox = new GtkHBox();
10 $hbox->pack_start(new GtkLabel(), true); // note 1
11 $hbox->pack_start($button, false); // note 2
12 $vbox->pack_start($hbox, false); // note 3
13
14 $window->show_all();
15 Gtk::main();
16 ?>
```

Listing 2.4.php

Explanation

1. Here our expandable box is an empty GtkLabel with `expand=true`. Note that you may also use an empty GtkHBox or GtkVBox. The effect is the same.
2. `expand=false` will ensure that the button does not expand horizontally in the hbox.
3. `expand=false` will ensure that the button does not expand vertically in the vbox.

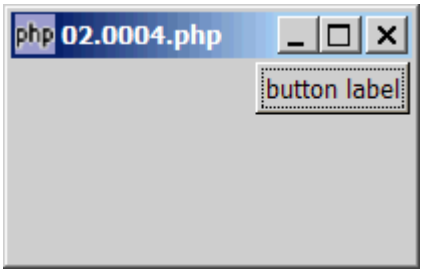
The net result is that the empty GtkLabel, because of its expandable spring, will "push" the button on the way to the right of the window, as the following diagram illustrates.



2.5 Center the button horizontally

Objective

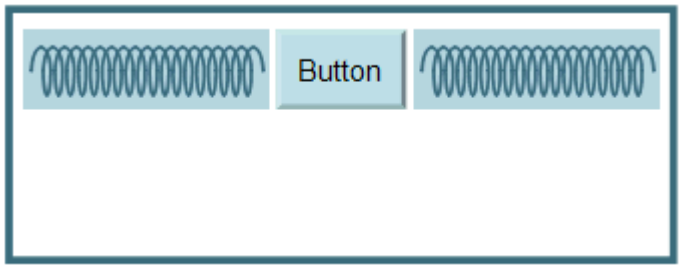
You have right aligned the button in the previous example as shown below:



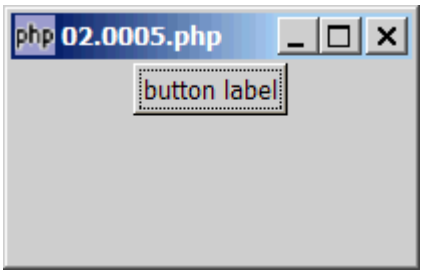
What if you want the button to be centered horizontally?

Overview

Instead of stuffing one spring box, we stuff two – one on each side as illustrated below:



Sample Output



Sample Code

```
1  <?php
2  $window = new GtkWindow();
3  $window->connect_simple('destroy',array('Gtk','main_quit'));
4  $window->set_size_request(200, 100);
5  $vbox = new GtkVBox();
6  $window->add($vbox);
7
```



```
8  $button = new GtkButton('button label');
9  $hbox = new GtkHBox();
10 $hbox->pack_start(new GtkHBox(), true); // note 1
11 $hbox->pack_start($button, false);
12 $hbox->pack_start(new GtkVBox(), true); // note 2
13
14 $vbox->pack_start($hbox, false);
15 $window->show_all();
16 Gtk::main();
17 ?>
```

Listing 2.5.php

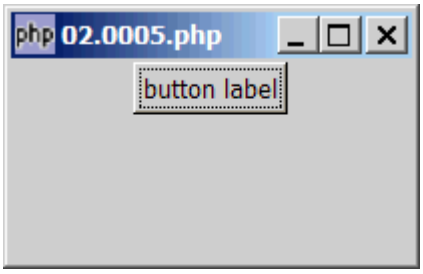
Explanation

1. This is the spring box on the left. Just for illustration sake, we use a hbox here to show you that the effect is the same as using an empty GtkLabel.
2. This is the spring box on the right. Just for illustration sake, we use a vbox here to show you that the effect is the same as using an empty GtkLabel.

2.6 Center the button horizontally and vertically

Objective

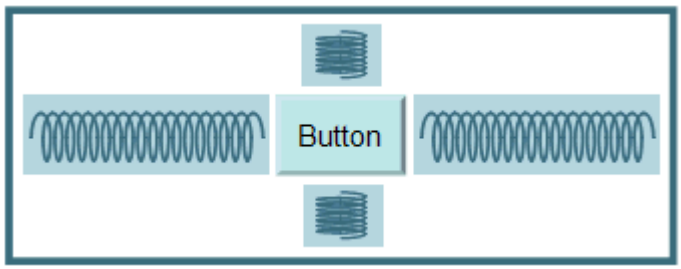
You have centered the button horizontally in the previous example as shown below:



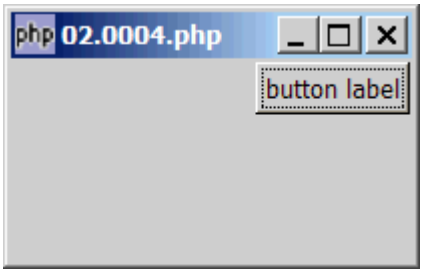
What if you want the button to be centered both horizontally and vertically?

Overview

In addition to the two spring boxes left and right, we stuff two more top and below too as illustrated below:



Sample Output



Sample Code

```
1  <?php
2  $window = new GtkWindow();
3  $window->connect_simple('destroy',array('Gtk','main_quit'));
4  $window->set_size_request(200, 100);
5  $vbox = new GtkVBox();
6  $window->add($vbox);
7
8  $button = new GtkButton('button label');
9  $hbox = new GtkHBox();
10 $hbox->pack_start(new GtkLabel(), true); // note 1
11 $hbox->pack_start($button, false);
12 $hbox->pack_start(new GtkLabel(), true); // note 1
13
14 $vbox->pack_start(new GtkLabel(), true); // note 2
15 $vbox->pack_start($hbox, false);
16 $vbox->pack_start(new GtkLabel(), true); // note 2
17 $window->show_all();
18 Gtk::main();
19 ?>
```

Listing 2.6.php

Explanation

1. The spring boxes left and right.
2. The spring boxes top and bottom.

2.7 Set the size of button

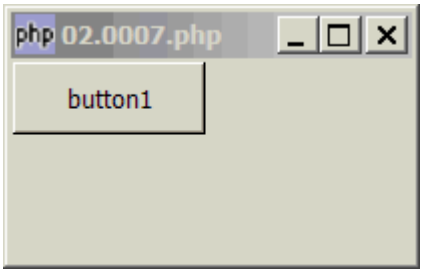
Objective

Suppose now you want the button to be exactly of the size 96 x 36.

Overview

- Set up to display the button in default size as described in Lesson 2.3.
- Set the size of the button with GtkWidget::set_size_request().

Sample Output



Sample Code

```
1  <?php
2  $window = new GtkWindow();
3  $window->connect_simple('destroy',array('Gtk','main_quit'));
4  $window->set_size_request(200, 100);
5  $vbox = new GtkVBox();
6  $window->add($vbox);
7
8  $button = new GtkButton('button1');
9  $button->set_size_request(96,36); // note 1
10 $hbox = new GtkHBox();
11 $hbox->pack_start($button, false); // note 2
12 $vbox->pack_start($hbox, false); // note 3
13
14 $window->show_all();
15 Gtk::main();
16 ?>
```

Listing 2.7.php

Explanation

1. Set the size of the button.
2. Stuff the button in hbox with `expand=false`.
3. Stuff the hbox in vbox with `expand=false`.

2.8 Have 3 buttons of size 60x36 at top left-hand corner

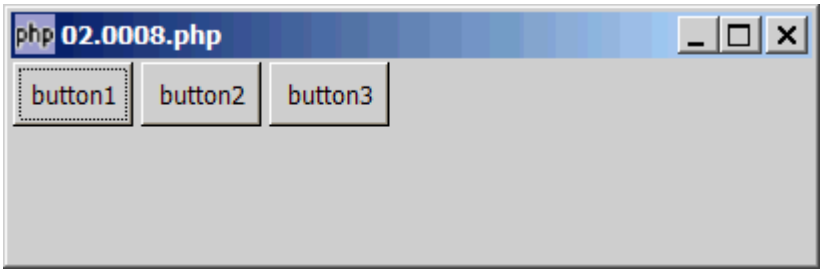
Objective

Suppose now you want to have three buttons at the top left-hand corner of the window, all of the size 60 x 36.

Overview

The concept is exactly the same as explained in the previous article. Instead of one button, we just continuously add three buttons using a for-next loop, each time packing the button into the hbox with `expand` set to `false`.

Sample Output



Sample Code

```
1  <?php
2  $window = new GtkWindow();
3  $window->connect_simple('destroy',array('Gtk','main_quit'));
4  $window->set_size_request(400, 100);
5  $vbox = new GtkVBox();
6  $window->add($vbox);
7
8  $hbox = new GtkHBox();
9  for ($i=1; $i<=3; ++$i) {
10    $button = new GtkButton('button'.$i);
11    $button->set_size_request(60,32); // note 1
12    $button->connect('clicked', 'on_click'); // note 2
13    $hbox->pack_start($button, false); // note 3
14    $hbox->pack_start(new GtkLabel(), false); // note 4
15 }
16
17 $vbox->pack_start($hbox, false);
18 $window->show_all();
19 Gtk::main();
20
21 function on_click($button) {
22    $button_label = $button->get_label(); // note 5
23    echo "you have clicked the button: $button_label\n";
```

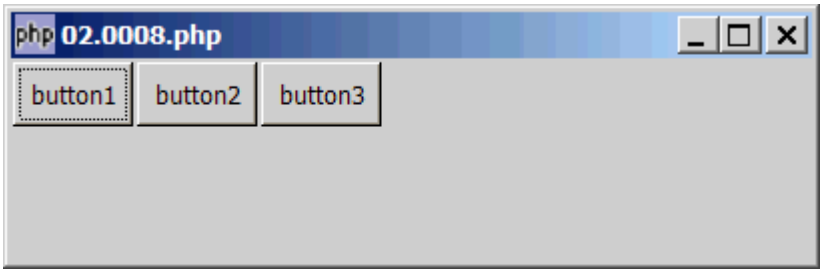


```
24 }
25 ?>
```

Listing 2.8.php

Explanation

1. Set the size of the button.
2. Set up the event handler for the button. We will explain in details about this in the chapter Events Handling.
3. Stuff the button in hbox with `expand=false`
4. This adds a little gap between the buttons. Try commenting out this line. You will get the following with all the button 'stacked' together.



5. Get the label of the button and echo it on the command window. We will explain in details about this in the chapter Events Handling.

2.9 Precise positioning of buttons

Objective

In the previous example, we have arbitrarily left a gap between the buttons.

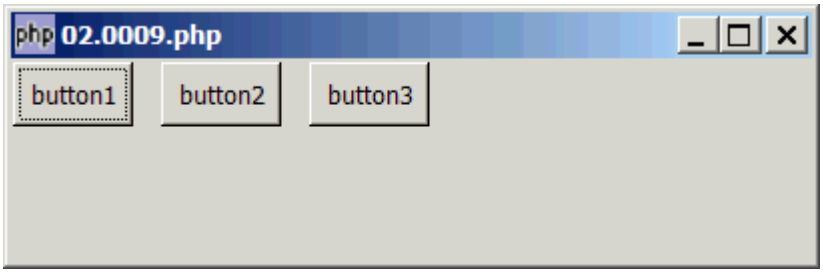
Suppose now you want to this gap to be exactly of 10 pixels wide between the buttons.

Overview

You might have read somewhere that it's not possible to do precise positioning in php-gtk, because everything is relative. The only way is to use GtkFixed but this is strongly discouraged because there will no longer be automatic layout management provided by php-gtk. This, is not really true.

If you understand the previous few examples, you already know how to do precise positioning and sizing in php-gtk! – yes, it is just a matter of stacking a pile of hboxes and vboxes on top of each other (just like the lego blocks), with the right setting of the expand parameter during packing.

Sample Output



Sample Code

```
1  <?php
2  $window = new GtkWindow();
3  $window->connect_simple('destroy',array('Gtk','main_quit'));
4  $window->set_size_request(400, 100);
5  $vbox = new GtkVBox();
6  $window->add($vbox);
7
8  $hbox = new GtkHBox();
9  for ($i=1; $i<=3; ++$i) {
10    $button = new GtkButton('button'.$i);
11    $button->set_size_request(60,32);
12    $button->connect('clicked', 'on_click');
13    $hbox->pack_start($button, false);
14
15    // add precisely a 10-pixel gap
16    $spacer = new GtkHBox(); // note 1
17    $spacer->set_size_request(10, -1); // note 2
18    $hbox->pack_start($spacer, false); // note 3
19 }
20
21 $vbox->pack_start($hbox, false);
22 $window->show_all();
23 Gtk::main();
24
25 function on_click($button) {
26    $button_label = $button->get_label();
27    echo "you have clicked the button: $button_label\n";
28 }
29 ?>
```

Listing 2.9.php

Explanation

We make use of the code in the previous example.

What's new here:

1. Create a new hbox. This hbox is the gap between the buttons.
2. Set the width of the hbox (i.e. the gap) to width=10. Since we are only concerned about the width here, we set the height to -1.
3. Now pack this gap into the outside hbox with `expand=false`.

Now try to resize the window. You will see that the gap will always remain precisely at 10 pixel because we have set `expand=false`.

2.10 Introducing the spacer

Objective

It's a good time now we introduce the spacer function, which you will find them useful in all the php-gtk applications that you'll be working on.

Overview

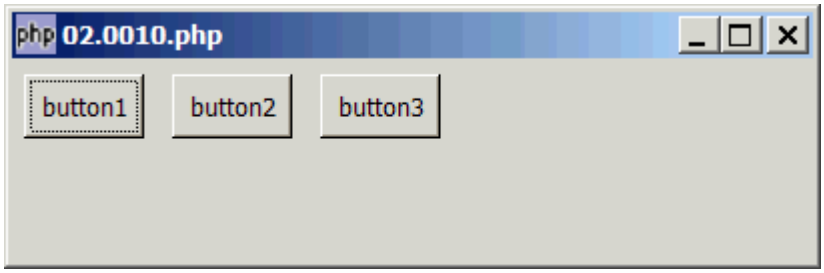
The idea of spacer should be very familiar to those people who have developed websites. Before CSS became popular, we usually use tables and "spacers" to achieve the desired layout. We use the spacer, which is just a small transparent image, set its width and height, to achieve proper layout and positioning.

If you have run through the previous example, you will find the technique to achieve precise positioning in php-gtk is almost entirely similar.

Since we use spacers so often in php-gtk, let's turn it into a function so that anytime we need to add a spacer, we just use a one-liner `spacer($hbox, 10)`.

Now that we have the spacer function, let's also conveniently use it to add a gap of 4 pixels so that the buttons are 4 pixels away from the top of the window. Let's also leave a gap of 3 pixels between the first button and the left side of the window so that it's aesthetically more pleasing to the eyes.

Sample Output



Sample Code

```
<?php
$window = new GtkWindow();
$window->connect_simple('destroy',array('Gtk','main_quit'));
$window->set_size_request(400, 100);
$vbox = new GtkVBox();
$window->add($vbox);

$hbox = new GtkHBox();
spacer($hbox, 3); // note 2
for ($i=1; $i<=3; ++$i) {
   $button = new GtkButton('button'.$i);
   $button->set_size_request(60,32);
   $button->connect('clicked', 'on_click');
   $hbox->pack_start($button, false);
   spacer($hbox, 10); // note 3
}

spacer($vbox, 4); // note 1
$vbox->pack_start($hbox, false);
$window->show_all();
Gtk::main();

function on_click($button) {
   $button_label = $button->get_label();
   echo "you have clicked the button: $button_label\n";
}

function spacer($container, $gap) { // note 4
   if ($container->get_name()=='GtkHBox') {
      echo "horizontal spacer: $gap\n";
      $spacer = new GtkHBox();
      $spacer->set_size_request($gap, -1);
      $container->pack_start($spacer, false);
   } else {
      echo "vertical spacer: $gap\n";
      $spacer = new GtkVBox();
      $spacer->set_size_request(-1, $gap);
      $container->pack_start($spacer, false);
   }
}
?>
```

Listing 2.10.php

Explanation

We make use of the code in previous example.

What's new here:

1. Leave a gap of 4 pixels between top of window and the buttons.
2. Leave a gap of 3 pixels between left of window and 1st button.
3. Leave a gap of 10 pixels between the buttons..
4. This is the spacer function. It just wraps the method as outlined in the previous example into a function. Note that GtkWidget::get_name() is a useful function that can be used to find out whether $container is a hbox or a vbox.

   For hbox, the gap will be horizontal. Hence we create a new hbox as the spacer and set its width using `$spacer->set_size_request($gap, -1)`.

   For vbox, the gap will be vertical. Hence we create a new vbox as the spacer and set its height using `$spacer->set_size_request(-1, $gap)`.

2.11 Introducing the expandable spacer

Objective

Now that we have the useful `spacer()` function, let's also create another very useful function – the `expandable_spacer()` function!

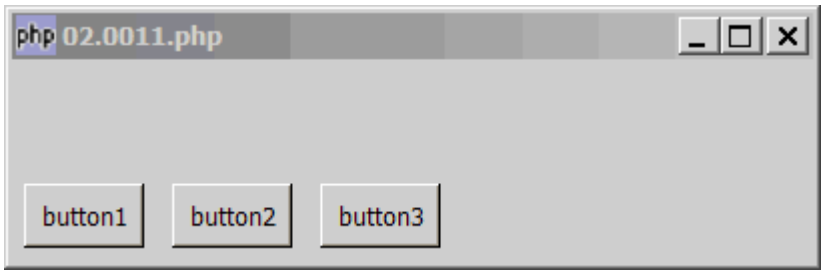
As an example on the use of `expandable_spacer()`, let's assume now you want the three buttons in the previous example to always stay at the bottom of the window.

Overview

This is the equivalent of the "spring box" which we used to right or center align the buttons in the earlier examples. The concept is very similar to a spacer, except this spacer has no fixed width or height. BUT, it has "springs" inside, always trying to grab whatever space it can get. So we will call this "spring box" an expandable spacer.

Note that for expandable spacer, you do not need to specify any width or height, since their width/height is dynamic. They will grab whatever space they can get when you do any resize of the windows.

Sample Output



Sample Code

```
<?php
$window = new GtkWindow();
$window->connect_simple('destroy',array('Gtk','main_quit'));
$window->set_size_request(400, 100);
$vbox = new GtkVBox();
$window->add($vbox);

$hbox = new GtkHBox();
spacer($hbox, 3);

for ($i=1; $i<=3; ++$i) {
   $button = new GtkButton('button'.$i);
   $button->set_size_request(60,32);
   $button->connect('clicked', 'on_click');
   $hbox->pack_start($button, false);
   spacer($hbox, 10);
}

expandable_spacer($vbox); // note 1
$vbox->pack_start($hbox, false);
spacer($vbox, 4); // note 2

$window->show_all();
Gtk::main();

function on_click($button) {
   $button_label = $button->get_label();
   echo "you have clicked the button: $button_label\n";
}

function spacer($container, $gap) {
   if ($container->get_name()=='GtkHBox') {
      echo "horizontal spacer: $gap\n";
      $spacer = new GtkHBox();
      $spacer->set_size_request($gap, -1);
      $container->pack_start($spacer, false);
   } else {
      echo "vertical spacer: $gap\n";
      $spacer = new GtkVBox();
      $spacer->set_size_request(-1, $gap);
      $container->pack_start($spacer, false);
   }
}

function expandable_spacer($container) { // note 3
   $container->pack_start(new GtkHBox(), true);
}
```



```
48
49 ?>
```

Listing 2.11.php

Explanation

We make use of the code in previous example.

What's new here:

1. Insert an *expandable spacer* at the top of the vbox so that the buttons will be pushed to stay at the bottom of the window.
2. As in the previous example, let's leave a gap of 3 pixels between left of window and first button.
3. This is the *expandable spacer* function. Note that this is just a one-liner! As explained in the previous examples, all we need here is a empty rectangular widget to act as a spring box. We used GtkHBox here. But you could replace this with a GtkVBox, or an empty GtkLabel.

Try to resize the window. You will see that the expandable_spacecr will automatically expand such that the three buttons will always be pushed to stay at the bottom of the window.

2.12 Add a Quit button that always stay at top right-hand corner

Objective

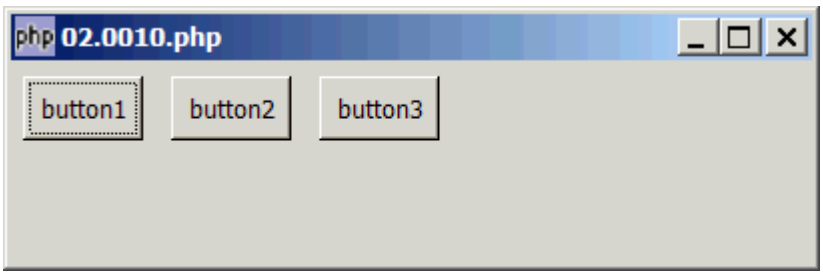
Let's have one more example on the combined use of spacer and expandable_spacer.

Suppose now you want to add a Quit button such that this button will always stay at the top right-hand corner, no matter how the user resize the window.

Overview

We just need to insert an expandable spacer between the third button and the Quit button.

Sample Output



Sample Code

```
<?php
$window = new GtkWindow();
$window->connect_simple('destroy',array('Gtk','main_quit'));
$window->set_size_request(400, 100);
$vbox = new GtkVBox();
$window->add($vbox);

$hbox = new GtkHBox();
spacer($hbox, 3);
for ($i=1; $i<=3; ++$i) {
   $button = new GtkButton('button'.$i);
   $button->set_size_request(60,32);
   $button->connect('clicked', 'on_click');
   $hbox->pack_start($button, false);
   spacer($hbox, 10);
}

// Add a quit button
$button = new GtkButton('Quit');
$button->set_size_request(60,32);
$button->connect('clicked', 'on_click', $button);
expandable_spacer($hbox); // note 1
$hbox->pack_start($button, false);
spacer($hbox, 3); // note 2

spacer($vbox, 4);
$vbox->pack_start($hbox, false);
$window->show_all();
Gtk::main();

function on_click($button) {
   $button_label = $button->get_label();
   echo "you have clicked the button: $button_label\n";
   if ($button_label=='Quit') Gtk::main_quit(); // note 3
}

function spacer($container, $gap) {
   if ($container->get_name()=='GtkHBox') {
      echo "horizontal spacer: $gap\n";
      $spacer = new GtkHBox();
      $spacer->set_size_request($gap, -1);
      $container->pack_start($spacer, false);
   } else {
      echo "vertical spacer: $gap\n";
      $spacer = new GtkVBox();
      $spacer->set_size_request(-1, $gap);
      $container->pack_start($spacer, false);
   }
```



```
49 }
50
51 function expandable_spacer($container) { // note 3
52     $container->pack_start(new GtkHBox(), true);
53 }
54
55 ?>
```

Explanation

We make use of the code in previous example.

What's new here:

1. Insert an expandable spacer between the Button3 and the Quit button.
2. Just like the first button, we also leave a 3-pixel gap between the Quit button and the right margin.
3. Exit the application if user clicks the Quit button.

Try to resize the window. You will see that the Quit button will always stay at the top right-hand corner of the window.

2.13 A simple form with only one entry field

Objective

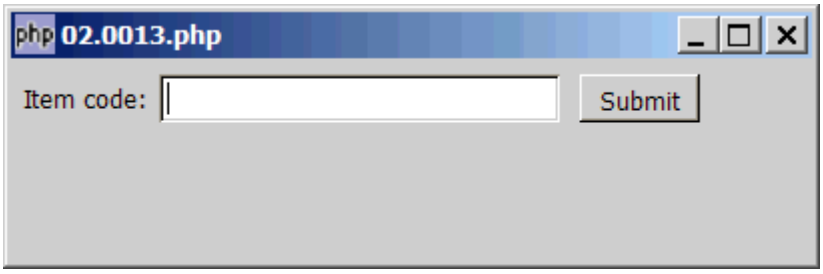
A form is very common in php-gtk applications. Each field usually has a label plus an entry box for user to enter inputs. We will now learn how to display such entry fields. For this example, we will start with only one field.

Overview

Until now we have only used GtkLabel and GtkButton. In this example, we will use one more widget, the GtkEntry, for data input.

- Create a GtkLabel.
- Create the GtkEntry.
- Create a GtkButton as the submit button.

Sample Output



Sample Code


```
<?php
$window = new GtkWindow();
$window->connect_simple('destroy',array('Gtk','main_quit'));
$window->set_size_request(400, 100);
$vbox = new GtkVBox();
$window->add($vbox);

$hbox = new GtkHBox();

spacer($hbox, 3); // add a 3-pixel left margin

// the label
$hbox->pack_start(new GtkLabel('Item code: '), false);

// the entry field
$input = new GtkEntry();
$input->set_size_request(200, -1);
$hbox->pack_start($input, false);

// the submit button
$button = new GtkButton('Submit');
$button->set_size_request(60,24);
$button->connect('clicked', 'on_submit', $input); // note 3
spacer($hbox, 6); // note 1
$hbox->pack_start($button, false);

spacer($vbox, 4); // add a 4-pixel top margin
$vbox->pack_start($hbox, false);

$window->show_all();
Gtk::main();

function on_submit($button, $input) { // note 3
   $str = $input->get_text(); // note 2
   echo "you have entered: $str\n";
}

function spacer($container, $gap) {
   if ($container->get_name()=='GtkHBox') {
      $spacer = new GtkHBox();
      $spacer->set_size_request($gap, -1);
   } else {
      $spacer = new GtkVBox();
      $spacer->set_size_request(-1, $gap);
   }
   $container->pack_start($spacer, false);
}

function expandable_spacer($container) {
```

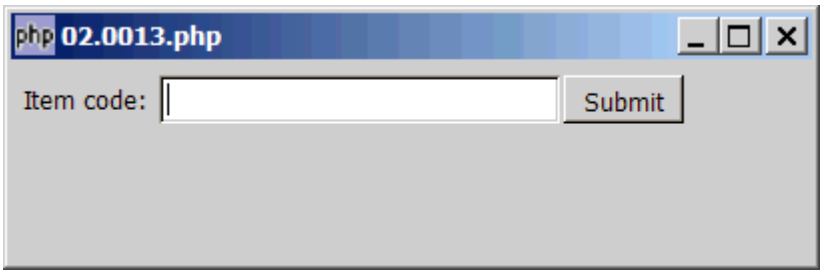


```
50     $container->pack_start(new GtkHBox(), true);
51 }
52
53 ?>
```

Listing 2.13.php

Explanation

1. Try commenting this line. You will find that without the spacer, the submit button will stick to the input field as follows:



2. This is how we get the user input from an entry field with the use of the method GtkEntry::set_text().
3. Note how the pointer to the GtkEntry `$input` is passed along with the signal handler. We need this to retrieve the value input by the user. We will explain this in more detail in the next Chapter on Signal Handling.

2.14 A form with three fields - Part 1

Objective

In this example, we will expand the form to include three fields. Take a look at the sample output. Yes, I know the layout is not right, and the button is too wide. And you may say that why not do this with GtkTable?

Well, be patient. I just want to show you that with simple widgets like GtkHBox and GtkVBox, we can accomplish the same effect as provided by some other widgets such as GtkTable and GtkIconView – with the added benefit that you have a lot more control over the positioning and sizing.

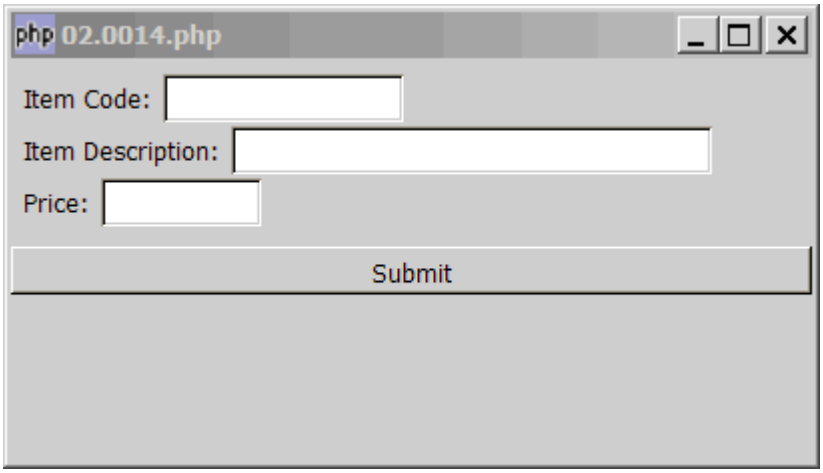
Go through these couple of exercises with me. Many of the techniques used here will be applicable to other widgets such as GtkTable.

Overview

We use exactly the same technique as described in the previous example:

- Each field will occupy one row. So we create a new hbox for each field.
- For each field, we create a GtkLabel and a GtkEntry, set the desired width, and pack them into the hbox – all with `expand=false`.

Sample Output



Sample Code

```
1  <?php
2  $window = new GtkWindow();
3  $window->connect_simple('destroy',array('Gtk','main_quit'));
4  $window->set_size_request(400, 200);
5  $vbox = new GtkVBox();
6  $window->add($vbox);
7  spacer($vbox, 4); // add a 4-pixel top margin
8
9  $input_field_def = array( // note 1
10    'Item Code'=>120,
11    'Item Description'=>240,
12    'Price'=>80);
13
14 foreach($input_field_def as $label=>$field_width) {
15    $hbox = new GtkHBox(); // note 2
16    spacer($hbox, 3); // add a 3-pixel left margin
17
18    // the label
19    $hbox->pack_start(new GtkLabel("$label: "), false);
20
21    // the entry field
22    $input = new GtkEntry();
23    $input->set_size_request($field_width, -1);
24    $hbox->pack_start($input, false);
25
26    $vbox->pack_start($hbox, false);
27 }
28
29 // the submit button
30 $button = new GtkButton('Submit');
31 $button->set_size_request(60,24);
```



```
32 $button->connect('clicked', 'on_submit', $input);
33 spacer($vbox, 6); //add a 6-pixel gap
34 $vbox->pack_start($button, false);  // note 3
35
36 $window->show_all();
37 Gtk::main();
38
39 function on_submit($button, $input) {
40    $str = $input->get_text();
41    echo "you have entered: $str\n";
42 }
43
44 function spacer($container, $gap) {
45    if ($container->get_name()=='GtkHBox') {
46       $spacer = new GtkHBox();
47       $spacer->set_size_request($gap, -1);
48    } else {
49       $spacer = new GtkVBox();
50       $spacer->set_size_request(-1, $gap);
51    }
52    $container->pack_start($spacer, false);
53 }
54
55 function expandable_spacer($container) {
56    $container->pack_start(new GtkHBox(), true);
57 }
58
59 ?>
```

Listing 2.14.php

Explanation

We make use of the code in the previous example.

What's new here:

1. This array defines the fields. Each associative array comprises "label_of_field" => width_of_field.
2. Create a new row for each field.
3. Add a submit button after all the fields are displayed.

2.15 A form with three fields - Part 2

Objective

In the previous example, we have displayed a form with three fields. However, the layout is not right, and the button is too wide.

We'll fix this in this example.

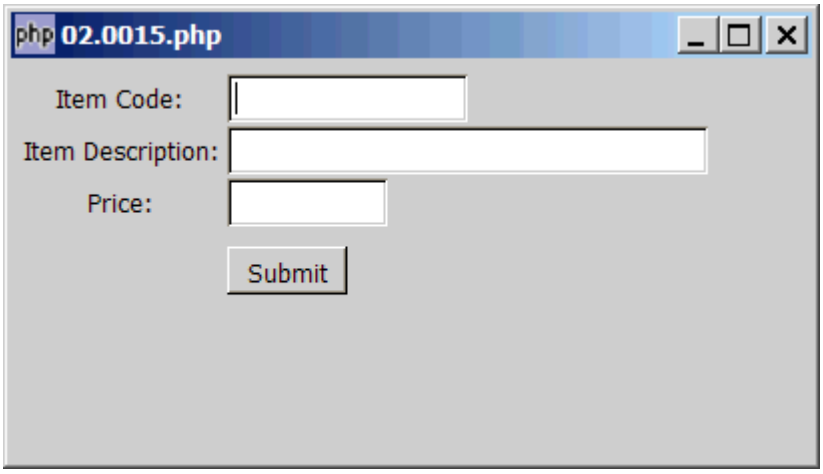
Overview

We use exactly the same technique as described in the previous example:

- To have a better layout, we set the width of all the labels to the same size with GtkWidget::set_size_request().
- We use the technique as described in Lesson 2.7 Set the size of button to set the Submit button to the right size.

Sample Output

Notice that we have fixed the layout, and the button size is correct. However, the label is centered. It should be left or right-justified. We'll fix this in the next article.



Sample Code

```
1  <?php
2  $window = new GtkWindow();
3  $window->connect_simple('destroy',array('Gtk','main_quit'));
4  $window->set_size_request(400, 200);
5  $vbox = new GtkVBox();
6  $window->add($vbox);
7  spacer($vbox, 4); // add a 4-pixel top margin
8
9  $input_field_def = array(
10    'Item Code'=>120,
11    'Item Description'=>240,
12    'Price'=>80);
13
14 foreach($input_field_def as $label=>$field_width) {
15    $hbox = new GtkHBox();
16    spacer($hbox, 3); // add a 3-pixel left margin
17
18    // the label
19    $label = new GtkLabel("$label: ");
```



```
20      $label->set_size_request(100, -1); // note 1
21      $hbox->pack_start($label, false);
22
23      // the entry field
24      $input = new GtkEntry();
25      $input->set_size_request($field_width, -1);
26      $hbox->pack_start($input, false);
27
28      $vbox->pack_start($hbox, false);
29  }
30
31  // the submit button
32  $hbox = new GtkHBox(); // note 2
33  $button = new GtkButton('Submit'); // note 2
34  $button->set_size_request(60,24); // note 2
35  $button->connect('clicked', 'on_submit', $input);
36  spacer($hbox, 3);
37  spacer($hbox, 100); // note 3
38  $hbox->pack_start($button, false);
39
40  spacer($vbox, 6); //add a 6-pixel gap
41  $vbox->pack_start($hbox, false);
42
43  $window->show_all();
44  Gtk::main();
45
46  function on_submit($button, $input) {
47      $str = $input->get_text();
48      echo "you have entered: $str\n";
49  }
50
51  function spacer($container, $gap) {
52      if ($container->get_name()=='GtkHBox') {
53          $spacer = new GtkHBox();
54          $spacer->set_size_request($gap, -1);
55      } else {
56          $spacer = new GtkVBox();
57          $spacer->set_size_request(-1, $gap);
58      }
59      $container->pack_start($spacer, false);
60  }
61
62  function expandable_spacer($container) {
63      $container->pack_start(new GtkHBox(), true);
64  }
65
66  ?>
```

Listing 2.15.php

Explanation

We make use of the code in the previous example.

What's new here:

1. Set the size of label to the same width.
2. Set the Submit button to the right size by packing it inside a hbox with `expand` set to `false`.
3. Note that we add a 100-pixel spacer here so that the Submit button appears right below the input fields.

Note

Note that all the labels are centered. This is the default setting of a GtkLabel.

2.16 A form with three fields - Part 3

Objective

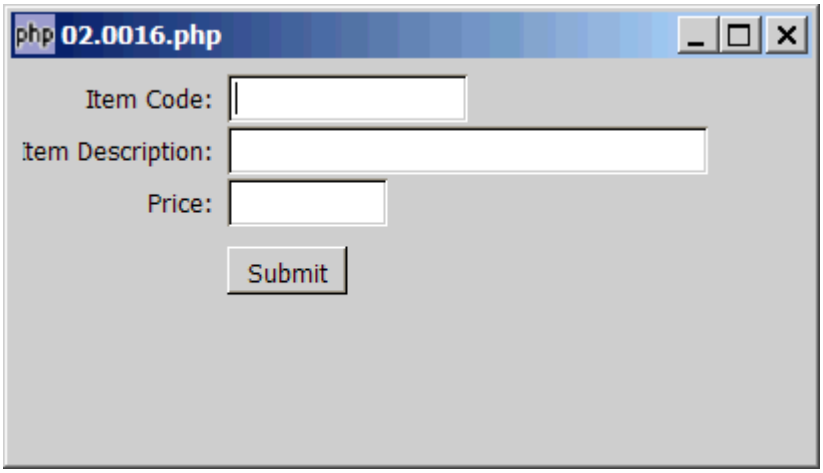
In the previous example, the field label is centered. We will fix this to make it right-justified.

Overview

GtkLabel, by default, centers the text. To make it left- or right-justified, just create a new hbox, pack it inside with `expand` set to `false`.

- If you want it left-justified, add an expandable spacer to its right.
- If you want it right-justified, add an expandable spacer to its left.

Sample Output



Sample Code

```
<?php
$window = new GtkWindow();
$window->connect_simple('destroy',array('Gtk','main_quit'));
$window->set_size_request(400, 200);
$vbox = new GtkVBox();
$window->add($vbox);
spacer($vbox, 4); // add a 4-pixel top margin

$input_field_def = array(
   'Item Code'=>120,
   'Item Description'=>240,
   'Price'=>80);

foreach($input_field_def as $label=>$field_width) {
   $hbox = new GtkHBox();
   spacer($hbox, 3); // add a 3-pixel left margin

   // the label
   $label = new GtkLabel("$label: ");
   $label_box = new GtkHBox(); // note 1
   $label_box->set_size_request(100, -1); // note 2
   expandable_spacer($label_box); // note 3
   $label_box->pack_start($label, false); // note 4
   $hbox->pack_start($label_box, false);

   // the entry field
   $input = new GtkEntry();
   $input->set_size_request($field_width, -1);
   $hbox->pack_start($input, false);

   $vbox->pack_start($hbox, false);
}

// the submit button
$hbox = new GtkHBox();
$button = new GtkButton('Submit');
$button->set_size_request(60,24);
$button->connect('clicked', 'on_submit', $input);
spacer($hbox, 3);
spacer($hbox, 100);
$hbox->pack_start($button, false);

spacer($vbox, 6); //add a 6-pixel gap
$vbox->pack_start($hbox, false);

$window->show_all();
Gtk::main();
```



```
48
49 function on_submit($button, $input) {
50    $str = $input->get_text();
51    echo "you have entered: $str\n";
52 }
53
54 function spacer($container, $gap) {
55    if ($container->get_name()=='GtkHBox') {
56       $spacer = new GtkHBox();
57       $spacer->set_size_request($gap, -1);
58    } else {
59       $spacer = new GtkVBox();
60       $spacer->set_size_request(-1, $gap);
61    }
62    $container->pack_start($spacer, false);
63 }
64
65 function expandable_spacer($container) {
66    $container->pack_start(new GtkHBox(), true);
67 }
68
69 ?>
```

Listing 2.16.php

Explanation

We make use of the code in the previous example.

What's new here:

1. Create a new hbox to hold the label.
2. Set the hbox size to the desired width.
3. We want the label to be right-justified in this example. So we add an expandable spacer first.
4. Then we pack the label into the hbox with expand=false.

Note

If you have used GtkTable before, you will realize that by default the contents are also displayed centered. You can use the same technique as described above to perform left- or right-justification of the contents.

2.17 Summary

Here's a quick summary of what we have learned in this chapter:

- With proper use of GtkHBox, GtkVBox, and a good understanding of the *Expand* and *Fill* parameter, you can achieve precise positioning, layout and sizing of the widgets in php-gtk2.
- We introduced the **spring box model** that would allow you to easily "visualize" the effect of the `expand` parameter.
- Whenever `set_size_request()` does not seem to have any effect, pack them into a GtkHBox, followed by a GtkVBox, with `expand` set to `false`.
- We introduced two useful functions: `spacer()` and `expandable_spacer()`.
- Use `spacer()` for precise positioning e.g. 4 pixel gap between buttons.
- Use `expandable_spacer()` when you want to do left/right justification, or you want a widget to stay on the top or bottom of window.

More Examples

To see how some of these techniques are being used in practice, you may refer to the following examples in the php-gtk2 Cookbook website:

- How to display a 2D array in table - Part 4?
- How to set valign top in GtkTable?
- How to set align left and valign top in GtkTable - using vbox and hbox?
- How to put a clickable link in GtkLabel - Part 2?
- How to display a popup alert for required fields - Part 1?
- How to display a popup alert for required fields - Part 2 (OK button centered)?
- How to display a popup dialog to prompt for data?
- How to allow only numbers in GtkEntry - Part 2?
- How to run multiple applications in multiple windows - Part 1?

Chapter 3 Signal Handling

What are Signals

Signals are the links between your php-gtk application and the user. Signals allow you to know when the user do something — clicked a button, pressed a key, inserted a character, deleted a row, selected an image, etc. Through signals, you are able to act accordingly based on user's actions.

Connecting Signals

Signals are like the newsletters or news feeds. There are millions of free newsletters and news feeds currently out there on Internet. You only subscribe to the ones you are interested, so that you will not be flooded with too much information.

Same for signals. Take a look at the php-gtk manual. There are currently 336 signals listed there. (Php-gtk2 is still under development, so more signals are continuously being added.) You probably do not need to be informed about all these signals in your application. You register with php-gtk which are the ones you're interested in with the `connect()` method.

Signal Name

Take a look again at the index of signals in the php-gtk manual. In php-gtk each signal is identified by a unique name. For example,

- We have seen that the signal that is emitted when the user clicks the button is called 'clicked'
- The signal that is emitted when the button is being pressed is called 'pressed'.
- The signal that is emitted when the button is being released is called 'pressed'.

When you register a signal with `connect()`, you need to know the name of this signal and specify it as the first argument.

Callback Function

When you register a signal with `connect()`, you also specify the name of your callback function as the second argument. For example, after you have registered the 'click' signal with `$button->connect('clicked', 'on_click')`, every time the user clicks the button, php-gtk will automatically call your function `on_click()` where you can perform your necessary action.

Since the callback function is used to handle a signal, you will sometimes see a callback function being referred to as a *signal handler*.

3.1 Signal basics

Objective

Let's begin our understanding of signal handling with the button click example.

Overview

- We "inform" php-gtk that we want to listen to a particular signal with `connect(signal, callback_fn)`.
- And we write the function that will be used to handle this signal. This function is called the *callback function* or *signal handler*.

Sample Output

Sample Code

```
1  <?php
2  $window = new GtkWindow();
3  $window->connect_simple('destroy',array('Gtk','main_quit'));
4  $window->set_size_request(200, 100);
5  $vbox = new GtkVBox();
6  $window->add($vbox);
7
8  $button = new GtkButton('button1');
9  $hbox = new GtkHBox();
10 $hbox->pack_start($button, true, false);
11 $vbox->pack_start($hbox, true, false);
12
13 $button->connect('clicked', 'on_click'); // note 1
14
15 $window->show_all();
16 Gtk::main();
17
18 function on_click($button) { // note 2
19    echo "button clicked!\n";
20 }
21 ?>
```

Listing 3.1.php

Explanation

1. The first parameter is the signal we want to listen to. In this example, we want to know when the button is clicked. The name of this signal is called 'clicked'. The second parameter is the name of your callback function that will be used to handle the signal. In this example, this is the function '`on_click()`'.
2. This is the callback function that is used to handle the signal 'clicked', i.e. when the user clicks the button, php-gtk will automatically calls this function that you have written. For this example, we simply echo the string "button clicked" to the command window.

3.2 Handling three buttons with one signal handler

Objective

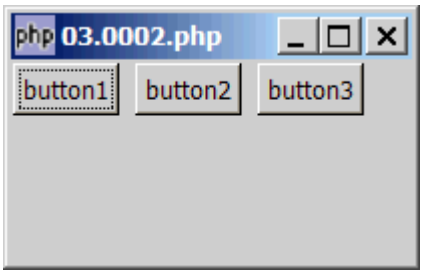
In the previous example, we have one button, and one callback function to handle its button clicks.

Suppose now we have three buttons. Of course, we could connect the button clicks to three separate callback functions. However, php-gtk also allows us to handle all the button clicks with just one signal handler.

Overview

- connect the signal 'clicked' for all the three buttons to the same callback function.
- In the callback function, you will find that the first argument $button gives you a pointer to the button that is being clicked on.
- To retrieve the label of the button, we use GtkButton::get_label().

Sample Output



Sample Code

```
1  <?php
2  $window = new GtkWindow();
3  $window->connect_simple('destroy',array('Gtk','main_quit'));
4  $window->set_size_request(200, 100);
5  $vbox = new GtkVBox();
6  $window->add($vbox);
```



```
7
8  $hbox = new GtkHBox();
9  for ($i=1; $i<=3; ++$i) {
10    $button = new GtkButton('button'.$i);
11    $button->connect('clicked', 'on_click'); // note 1
12    $hbox->pack_start($button, false);
13    $hbox->pack_start(new GtkLabel(' '), false);
14 }
15
16 $vbox->pack_start($hbox, false);
17 $window->show_all();
18 Gtk::main();
19
20 function on_click($button) { // note 2
21    $button_label = $button->get_label(); // note 3
22    echo "you have clicked the button: $button_label\n";
23 }
24 ?>
```

Listing 3.2.php

Explanation

1. Connect the signal 'clicked' to the function on_click().
2. Note the first argument $button. This is passed along by php-gtk when calling your signal handler.
3. Get the label of the button that is being clicked on, and echo it in the command window.

Data Automatically Passed to Callback Functions

When php-gtk invokes your callback function, most of the time it will automatically pass along some useful data in the arguments. Take a look at some of these signals in the php-gtk manual:

- signal 'clicked': `void callback(GtkButton button)`
- signal 'key-press-event': `bool callback(GtkWidget widget, GdkEvent event)`
- signal 'set_cell_data_func': `void callback(GtkCellLayout cell_layout, GtkCellRenderer cell, GtkTreeModel tree_model, GtkTreeIter iter [, user_data])`

Note that:

- For each signal, different data are passed along by php-gtk.
- The first argument is usually the widget that emitted the signal.

Differentiating the Source of Signals

For the signal 'clicked', the first argument we receive in the callback function is a pointer to the GtkButton that is being clicked on. With this pointer, we make a call to

the method GtkButton::get_label() to get the corresponding label of the button. Therefore, even though we have used just one callback function, we're able to differentiate exactly which is the button that triggered the signal.

3.3 Handling multiple signals with one signal handler – one more example

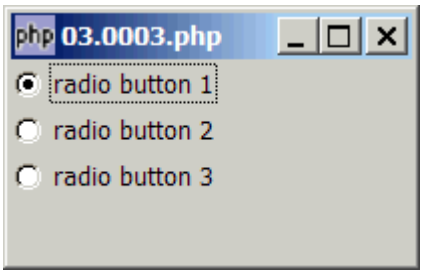
Objective

Let's have one more example of handling multiple signals using only one signal handler. This time instead of GtkButton, we will use GtkRadioButton.

Overview

- To display radio buttons, use GtkRadiobutton.
- When the user selects one of the radio buttons, the signal emitted is called 'toggled'.
- As in the previous example, we connect this signal for all the three radio buttons to the same callback function `on_toggle()`.
- To retrieve the label of the button, we use GtkButton::get_label(), where `$radio` is the first argument being passed along by php-gtk when the callback function is activated.

Sample Output



Sample Code

```
1  <?php
2  $window = new GtkWindow();
3  $window->connect_simple('destroy',array('Gtk','main_quit'));
4  $window->set_size_request(200, 100);
5  $vbox = new GtkVBox();
6  $window->add($vbox);
7
8  $radio = null;
9  for ($i=1; $i<=3; ++$i) {
10    $radio = new GtkRadioButton($radio, 'radio button '.$i); // note 1
11    $radio->connect('toggled', 'on_toggle'); // note 2
```



```
12      $vbox->pack_start($radio, false);
13  }
14
15  $window->show_all();
16  Gtk::main();
17
18  function on_toggle($radio) { // note 3
19      $label = $radio->get_label(); // note 4
20      $active = $radio->get_active(); // note 5
21      if ($active) echo "radio button pressed: $label\n";
22  }
23  ?>
```

Listing 3.3.php

Explanation

1. Create the radio buttons using `new GtkRadioButton(buttongrp, button_label)`. Note that for the first radio button, we pass in `NULL` as the first argument. For the second radio buttons, we pass in the pointer of the first radio button as the first argument. Similarly for the third. PHP-GTK will then group all these into the same group.
2. Connect the signal 'toggled' to the function `on_toggle()`.
3. Note the first argument `$radio`. This is passed along by php-gtk when calling your signal handler.
4. Get the label of the button that is being clicked on.
5. Check if the radio button is selected using the method GtkTogglebutton::get_active().

3.4 Passing additional data to callback function - Part 1

Objective

In HTML, we can use

```
<input type="radio" name="radiogrp1" value="NY">New York
```

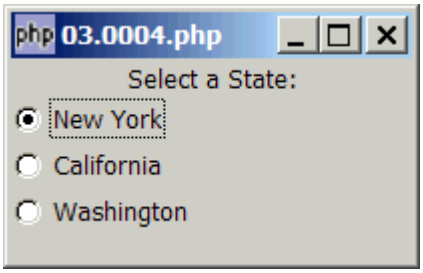
to attach a value to each radio item. When the user select, say New York, the value returned will be NY.

We will achieve the same effect using php-gtk in this example.

Overview

- Set up the radio buttons as outlined in the previous example.
- Specify any additional data to be passed along with the signal after the second argument in the connect statement.
- Note that your callback function definition should now contain the corresponding additional parameters to receive the additional data.

Sample Output



Sample Code

```
1  <?php
2  $window = new GtkWindow();
3  $window->connect_simple('destroy',array('Gtk','main_quit'));
4  $window->set_size_request(200, 100);
5  $vbox = new GtkVBox();
6  $window->add($vbox);
7  $vbox->pack_start(new GtkLabel('Select a State:'), false);
8
9  $radio = null;
10 $radio_button_def = array('New York'=>'NY', 'California'=>'CA', 'Washington'=>'WA');
11 foreach ($radio_button_def as $state_name => $state_code) {
12     $radio = new GtkRadioButton($radio, $state_name);
13     $radio->connect('toggled', 'on_toggle', $state_code); // note 1
14     $vbox->pack_start($radio, false);
15 }
16
17 $window->show_all();
18 Gtk::main();
19
20 function on_toggle($radio, $user_data) { // note 2
21     $label = $radio->get_label();
22     $active = $radio->get_active();
23     if ($active) echo "radio button pressed: $label ($user_data)\n";
24 }
25 ?>
```

Listing 3.4.php

Explanation

1. By specifying `$state_code` as the third argument, the 2-letter state code will also be passed to your callback function by php-gtk.
2. The first parameter is the widget that emitted the signal. This is passed along automatically by php-gtk. The second parameter `$user_data` is the additional data that you have specified in the connect statement.

3.5 Passing additional data to callback function - Part 2

Objective

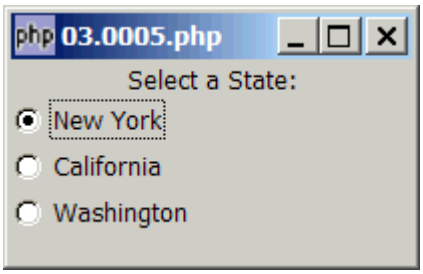
The additional data to be passed to the callback function can be of any valid PHP data type, e.g. integer, float, string, array, object, etc.

This example achieves the same effect as the previous example except that we pass along an *array* instead of a string.

Overview

- Use previous example as the base.
- In the connect statement, specify the array $radio_button_def instead of $state_code as the third argument.
- Change the second parameter of the callback function definition accordingly.

Sample Output



Sample Code

```
1  <?php
2  $window = new GtkWindow();
3  $window->connect_simple('destroy',array('Gtk','main_quit'));
4  $window->set_size_request(200, 100);
5  $vbox = new GtkVBox();
6  $window->add($vbox);
7  $vbox->pack_start(new GtkLabel('Select a State:'), false);
8
9  $radio = null;
10 $radio_button_def = array('New York'=>'NY', 'California'=>'CA', 'Washington'=>'WA');
11 foreach ($radio_button_def as $state_name => $state_code) {
12     $radio = new GtkRadioButton($radio, $state_name);
13     $radio->connect('toggled', 'on_toggle', $radio_button_def); // note 1
14     $vbox->pack_start($radio, false);
15 }
16
17 $window->show_all();
18 Gtk::main();
19
```



```
20 function on_toggle($radio, $button_def) { // note 2
21     $label = $radio->get_label();
22     $active = $radio->get_active();
23     $code = $button_def[$label]; // note 3
24     if ($active) echo "radio button pressed: $label ($code)\n";
25 }
26 ?>
```

Listing 3.5.php

Explanation

1. Pass the array `$radio_button_def` along with the signal.
2. The first parameter is the widget that emitted the signal. This is passed along automatically by php-gtk. The second parameter `$button_def` is the additional data that you have specified in the connect statement.
3. Get the 2-letter state code from the array, using `$label` as the index.

3.6 Passing additional data to callback function - Part 3

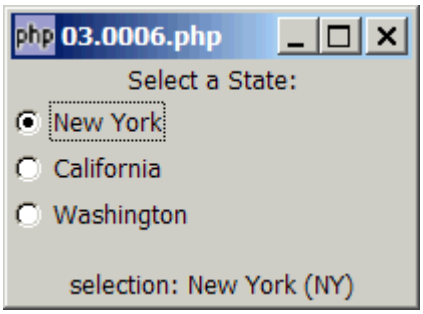
Objective

You can pass as many additional data as you want to the callback function along with the signal as illustrated by this example.

Overview

- Attach additional data in the connect statement after the second argument. Separate each data by commas.
- Change the parameters of the callback function definition accordingly to match the number of additional data that will be received.

Sample Output



Sample Code

```
1  <?php
2  $window = new GtkWindow();
3  $window->connect_simple('destroy',array('Gtk','main_quit'));
4  $window->set_size_request(200, 120);
5  $vbox = new GtkVBox();
6  $window->add($vbox);
7  $vbox->pack_start(new GtkLabel('Select a State:'), false);
8  $vbox_radio_buttons = new GtkVBox(); // note 1
9  $vbox->pack_start($vbox_radio_buttons, false);
10 $vbox->pack_start(new GtkVBox()); // note 2
11 $status = new GtkLabel(' '); // note 3
12 $vbox->pack_start($status, false);
13
14 $radio = null;
15 $radio_button_def = array('New York'=>'NY', 'California'=>'CA', 'Washington'=>'WA');
16 foreach ($radio_button_def as $state_name => $state_code) {
17    $radio = new GtkRadioButton($radio, $state_name);
18    $radio->connect('toggled', 'on_toggle', $radio_button_def, $status); // note 4
19    $vbox_radio_buttons->pack_start($radio, false);
20 }
21
22 $window->show_all();
23 Gtk::main();
24
25 function on_toggle($radio, $button_def, $status) { // note 5
26    $label = $radio->get_label();
27    $active = $radio->get_active();
28    $code = $button_def[$label];
29    if ($active) $status->set_text("selection: $label ($code)");  // note 6
30 }
31 ?>
```

Listing 3.6.php

Explanation

1. Create one more vbox to hold the radio buttons.
2. Add an expandable spacer to push the status bar to the bottom.
3. Create a GtkLabel to act as a status line.
4. Note that we pass two pieces of additional data along with the signal this time: the button definition array, as well as the GtkLabel object (the status line).
5. Note that our callback function now has 3 parameters – one passed automatically by php-gtk, and the other two by us.
6. With the pointer to the status line `$status` passed along together with the signal, we can now echo the user selection using GtkLabel::set_text().

3.7 Object-oriented connections

Objective

Suppose you prefer using object-oriented programming, this example shows you how to connect signals to methods in classes.

Overview

The way to connect signals to methods in classes is as follows:

```
connect($signal_name, array($object, $method_name),
    [$user_data1, $user_data2, $user_data3, ...]);
```

Sample Output

Sample Code

```
1  <?php
2
3  class App {
4
5     function __construct() {
6        $window = new GtkWindow();
7        $window->connect_simple('destroy',array('Gtk','main_quit'));
8        $window->set_size_request(200, 120);
9        $vbox = new GtkVBox();
10       $window->add($vbox);
11       $vbox->pack_start(new GtkLabel('Select a State:'), false);
12       $vbox_radio_buttons = new GtkVBox();
13       $vbox->pack_start($vbox_radio_buttons, false);
14       $vbox->pack_start(new GtkVBox());
15       $status = new GtkLabel(' ');
16       $vbox->pack_start($status, false);
17
18       $radio = null;
19       $radio_button_def = array('New York'=>'NY', 'California'=>'CA',
   'Washington'=>'WA');
20       $i = 0;
21       foreach ($radio_button_def as $state_name => $state_code) {
```



```
22          $radio = new GtkRadioButton($radio, $state_name);
23          $radio->connect('toggled', array($this, 'on_toggle'),
24             $radio_button_def, $status); // note 1
25          $vbox_radio_buttons->pack_start($radio, false);
26       }
27
28       $window->show_all();
29       Gtk::main();
30    }
31
32    function on_toggle($radio, $button_def, $status) {
33       $label = $radio->get_label();
34       $active = $radio->get_active();
35       $code = $button_def[$label];
36       if ($active) $status->set_text("selection: $label ($code)");
37    }
38 }
39
40 $app = new App;
41
42 ?>
```

Listing 3.7.php

Explanation

1. `$this` refers to the class itself. '`on_toggle`' is the method used to handle the signal.

Note

- You should now be able to understand the statement:

  ```
  $window->connect_simple('destroy', array('Gtk','main_quit'));
  ```

  PHP-GTK2 itself is entirely object-oriented based (as we shall see in the next chapter). So the above simply says, when the signal '`destroy`' is detected, call the function Gtk::main_quit(), which is a method in the Gtk class to exit a php-gtk program.

- connect_simple() is similar to connect(). The only difference is that you use connect_simple when you do not need to pass any additional info to your callback function.

- Trying changing the above statement to:

  ```
  $window->connect('destroy', array('Gtk','main_quit'));
  ```

  Run the program and close the window. You will see a warning message:

  ```
  PHP Warning: Gtk::main_quit() requires exactly 0 arguments.
  1 given in ...
  ```


This is because by default, `connect()` will pass the widget that triggered the signal as the first argument to your callback function. Hence the warning "1 argument given". To avoid this warning, we use `connect_simple()` here.

- Alternatively, if you really want to use connect(), you can change the program as follows:

```
1  <?php
2
3  class App {
4
5     function __construct() {
6        $window = new GtkWindow();
7        $window->connect('destroy', array($this, 'on_destroy'));
8        $window->set_size_request(200, 120);
9        $vbox = new GtkVBox();
10       $window->add($vbox);
11       $vbox->pack_start(new GtkLabel('Select a State:'), false);
12       $vbox_radio_buttons = new GtkVBox();
13       $vbox->pack_start($vbox_radio_buttons, false);
14       $vbox->pack_start(new GtkVBox());
15       $status = new GtkLabel(' ');
16       $vbox->pack_start($status, false);
17
18       $radio = null;
19       $radio_button_def = array('New York'=>'NY', 'California'=>'CA',
   'Washington'=>'WA');
20       $i = 0;
21       foreach ($radio_button_def as $state_name => $state_code) {
22          $radio = new GtkRadioButton($radio, $state_name);
23          $radio->connect('toggled', array($this, 'on_toggle'),
24             $radio_button_def, $status);
25          $vbox_radio_buttons->pack_start($radio, false);
26       }
27
28       $window->show_all();
29       Gtk::main();
30    }
31
32    function on_toggle($radio, $button_def, $status) {
33       $label = $radio->get_label();
34       $active = $radio->get_active();
35       $code = $button_def[$label];
36       if ($active) $status->set_text("selection: $label ($code)");
37    }
38
39    function on_destroy($widget) {
40       Gtk::main_quit();
41    }
42 }
```



```
43
44 $app = new App;
45
46 ?>
```

Listing 3.7.1.php

3.8 Callback methods in another class

Objective

You can connect signals to methods in other classes too, as shown in the example below.

Overview

Just change `$this` in the previous example to the object containing the callback methods.

Sample Output

Sample Code

```
1  <?php
2
3  class App {
4
5     function __construct($obj_b) {
6        $window = new GtkWindow();
7        $window->connect_simple('destroy',array('Gtk','main_quit'));
8        $window->set_size_request(200, 120);
9        $vbox = new GtkVBox();
10       $window->add($vbox);
11       $vbox->pack_start(new GtkLabel('Select a State:'), false);
12       $vbox_radio_buttons = new GtkVBox();
13       $vbox->pack_start($vbox_radio_buttons, false);
14       $vbox->pack_start(new GtkVBox());
15       $status = new GtkLabel(' ');
16       $vbox->pack_start($status, false);
```



```
17
18          $radio = null;
19          $radio_button_def = array('New York'=>'NY', 'California'=>'CA',
    'Washington'=>'WA');
20          $i = 0;
21          foreach ($radio_button_def as $state_name => $state_code) {
22             $radio = new GtkRadioButton($radio, $state_name);
23             $radio->connect('toggled', array($obj_b, 'on_toggle'),
24                $radio_button_def, $status); // note 1
25             $vbox_radio_buttons->pack_start($radio, false);
26          }
27
28          $window->show_all();
29          Gtk::main();
30       }
31
32    }
33
34    class Class_B {
35       function on_toggle($radio, $button_def, $status) {
36          $label = $radio->get_label();
37          $active = $radio->get_active();
38          $code = $button_def[$label];
39          if ($active) $status->set_text("selection: $label ($code)");
40       }
41    }
42
43    $obj_b = new Class_B();
44    $app = new App($obj_b);
45
46    ?>
```

Listing 3.8.php

Explanation

1. In this example, the callback method resides in Class_B, hence the construct `array($obj_b, 'on_toggle')`.

3.9 Manually generating a signal

Objective

Signals are NOT all generated by users. You can **manually** generate a signal for some of the widgets too!

You might have noticed that when running example 3.6, the first time you run the program, the first radio item (New York) is selected, but nothing is displayed in the status line.

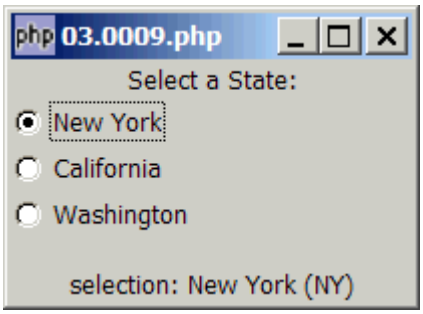
To fix this, we can manually write the selection in the status line on initialization of the program.

Another method is to manually generate a toggle signal, so that the status line is updated automatically by the callback function.

Overview

- Use the method GtkTogglebutton::toggled() to manually generate a 'toggled' signal.

Sample Output



Sample Code

```
1  <?php
2  $window = new GtkWindow();
3  $window->connect_simple('destroy',array('Gtk','main_quit'));
4  $window->set_size_request(200, 120);
5  $vbox = new GtkVBox();
6  $window->add($vbox);
7
8  $vbox->pack_start(new GtkLabel('Select a State:'), false);
9  $vbox_radio_buttons = new GtkVBox();
10 $vbox->pack_start($vbox_radio_buttons, false);
11 $vbox->pack_start(new GtkVBox());
12 $status = new GtkLabel(' ');
13 $vbox->pack_start($status, false);
14
15 $radio_button_def = array('New York'=>'NY', 'California'=>'CA', 'Washington'=>'WA');
16 $i = 0;
17 $radio[0] = null;
18 foreach ($radio_button_def as $state_name => $state_code) {
19    $radio[$i] = new GtkRadioButton($radio[0], $state_name); // note 1
20    $radio[$i]->connect('toggled', 'on_toggle', $radio_button_def, $status);
21    $vbox_radio_buttons->pack_start($radio[$i], false);
22    ++$i;
23 }
24 $radio[0]->toggled(); // note 2
25
```



```
26 $window->show_all();
27 Gtk::main();
28
29 function on_toggle($radio, $button_def, $status) {
30     $label = $radio->get_label();
31     $active = $radio->get_active();
32     $code = $button_def[$label];
33     if ($active) $status->set_text("selection: $label ($code)");
34 }
35 ?>
```

Listing 3.9.php

Explanation

1. Note that we changed the program slightly in the setting up of the radio buttons. This is because we need the pointer to the first radio button in order to manually generate a toggle.
2. Manually generate a toggle to the first radio button.

Note

Try running the above program. You will see that now the status line gets updated correctly when the program is first run.

3.10 Clickable label

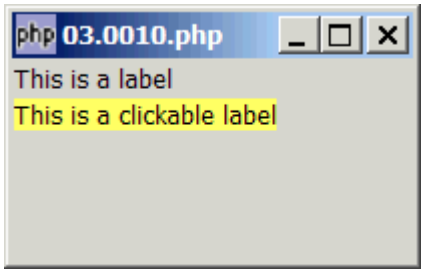
Objective

In this example, we will show you how to make a GtkLabel clickable.

Overview

- Create the GtkLabel.
- Create a new GtkEventBox.
- Stuff the label inside the eventbox.
- Register the signal 'button-press-event' on the eventbox.

Sample Output



Sample Code

```
1  <?php
2  $window = new GtkWindow();
3  $window->set_size_request(200, 100);
4  $window->connect_simple('destroy', array('Gtk','main_quit'));
5  $window->add($vbox = new GtkVBox());
6
7  // create a label
8  $hbox = new GtkHBox();
9  $hbox->pack_start(new GtkLabel("This is a label"), false);
10 $vbox->pack_start($hbox, false);
11
12 // create a clickable label
13 $eventbox = new GtkEventBox; // note 1
14 $eventbox->add(new GtkLabel("This is a clickable label")); // note 2
15 $eventbox->connect('button-press-event', 'on_button_press'); // note 3
16 $eventbox->modify_bg(Gtk::STATE_NORMAL, GdkColor::parse("#ffff66")); // note 4
17 $hbox = new GtkHBox();
18 $hbox->pack_start($eventbox, false);
19 $vbox->pack_start($hbox, false);
20
21 $window->show_all();
22 Gtk::main();
23
24 function on_button_press($widget, $event) { // note 5
25    echo "you have clicked the clickable label!\n";
26 }
27
28 ?>
```

Listing 3.10.php

Explanation

1. Create a new eventbox.
2. Stuff the label inside the eventbox. Note that GtkEventBox is a bin. It can only hold one widget. So we use GtkContainer::add() to add the widget, and not pack_start.
3. Connect the signal 'button-press-event' to the callback function on_button_press(). Note that the event is captured through the eventbox, not the label.
4. Set the background color of the clickable label.
5. This is the callback function that is called when the user clicks on the clickable label. Note the two arguments that are being passed to you by php-gtk. The first is the widget that generated the signal, in this case it's the clickable widget. The second is GdkEvent object. We will elaborate more about this in the next example.

Widgets without window

Some widget e.g. GtkLabel, does not have an associated window of its own. It draws on its parent's window. There are two major implications because of this:

- You cannot set its background color. It takes on the color of its parent's window.
- It cannot receive events e.g. receiving mouse clicks or keypress

Below are listed some of these widgets without window:

- GtkAlignment
- GtkArrow
- GtkAspectFrame
- GtkBin
- GtkBox
- GtkButton
- GtkCheckButton
- GtkFixed
- GtkFrame
- GtkHBox
- GtkHSeparator
- GtkImage
- GtkItem
- GtkLabel
- GtkMenuItem
- GtkNotebook
- GtkPaned
- GtkPixmap
- GtkRadioButton
- GtkRange
- GtkScrolledWindow
- GtkSeparator
- GtkTable
- GtkToolbar
- GtkVBox
- GtkVSeparator

Receiving events for widgets without window

If you want to be able to receive events (or set the background color) for those widgets without window, you can wrap these widgets inside an GtkEventBox as shown in the example above. Simply create a new GtkEventBox, and use add() to stuff the widget inside the eventbox. You can then receive mouse clicks or keypress through the eventbox.

3.11 Useful event properties from button-press-event

Objective

In the previous example, we have set up a 'button-press-event' to respond to the clickable label.

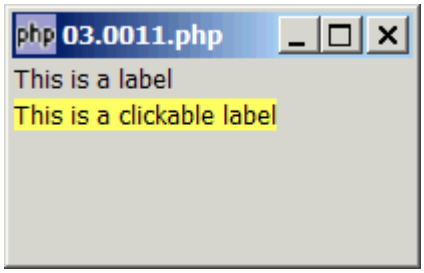
In this example, we will show you some of the useful event properties from this button-press-event.

Overview

In response to the signal 'button-press-event', the second argument passed to your callback function is a GdkEvent object. Some of the useful properties available from GdkEvent object are:

- `x, y`: x- and y-position of the mouse
- `button`: mouse button pressed 1=left, 2=middle, 3=right
- `state`: GdkModifierType - modifier key pressed e.g. ctrl, alt, shift, caps lock
- `type`: GdkEventType - event type e.g. single click, double click, triple click, etc.

Sample Output



Sample Code

```
1  <?php
2  $window = new GtkWindow();
3  $window->set_size_request(200, 100);
4  $window->connect_simple('destroy', array('Gtk','main_quit'));
5
6  $vbox = new GtkVBox();
7  $eventbox = new GtkEventBox;
```



```
8  $eventbox->add($vbox);
9  $window->add($eventbox);
10
11 // create a label
12 $hbox = new GtkHBox();
13 $hbox->pack_start(new GtkLabel("This is a label"), false);
14 $vbox->pack_start($hbox, false);
15
16 // create a clickable label
17 $eventbox = new GtkEventBox;
18 $eventbox->add(new GtkLabel("This is a clickable label"));
19 $eventbox->connect('button-press-event', 'on_button_press');
20 $eventbox->modify_bg(Gtk::STATE_NORMAL, GdkColor::parse("#ffff66"));
21 $hbox = new GtkHBox();
22 $hbox->pack_start($eventbox, false);
23 $vbox->pack_start($hbox, false);
24
25
26 $window->show_all();
27 Gtk::main();
28
29 function on_button_press($widget, $event) {
30    echo "you have clicked the clickable label!\n";
31    echo "x-pos = $event->x\n"; // note 1
32    echo "y-pos = $event->y\n"; // note 1
33
34    switch($event->button) { // note 2
35       case 1: echo "button = left\n"; break;
36       case 2: echo "button = middle\n"; break;
37       case 3: echo "button = right\n"; break;
38    }
39
40    $modifier = '';
41    if ($event->state & Gdk::SHIFT_MASK) $modifier .= '+shift'; // note 3
42    if ($event->state & Gdk::LOCK_MASK) $modifier .= '+lock'; // note 3
43    if( $event->state & Gdk::CONTROL_MASK) $modifier .= '+ctrl'; // note 3
44    if ($event->state & Gdk::MOD1_MASK) $modifier .= '+alt'; // note 3
45    if ($modifier!='') echo "modifier: ".substr($modifier, 1)."\n";
46
47    echo "type = $event->type\n";
48    switch($event->type) { // note 4
49       case Gdk::BUTTON_PRESS: echo "mouse single click\n"; break;
50       case Gdk::_2BUTTON_PRESS: echo "mouse double click\n"; break;
51       case Gdk::_3BUTTON_PRESS: echo "mouse triple click\n"; break;
52    }
53 }
54
55 ?>
```

Listing 3.11.php

Explanation

1. Get the x- and y- position of the mouse.
2. Which mouse button was pressed.
3. To test if the control key was pressed, we use `$event->state` & `Gdk::CONTROL_MASK`. The entire list of the modifier key names is here.
4. Test whether it's a single, double or triple mouse click. Take note that for a double click, a single click is also generated. Same for triple click.

3.12 Signal propagation

Objective

What confuses a lot of people new to php-gtk is how signals propagate across the different widgets. It's confusing because signals such as 'button-press-event' propagate differently than that of 'key-press-event'.

In this example, we will look at how 'button-press-event' propagates.

Overview

For most signals, such as the 'button-press-event', the signals are handled as follows:

- The widget where the event occurred will be the first to receive the signal.
- If the signal handler returns a TURE, all processing will stop.
- Otherwise, the signal will propagate to its parent.
- This will continue until it gets to the top-level widget if no one handles the event.

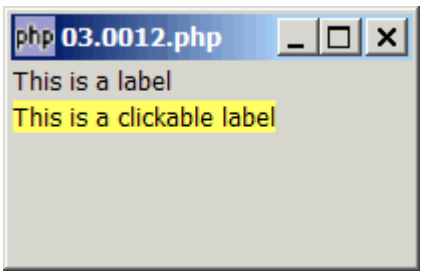
Sample Output

- Click on the clickable label. You should see two message lines displayed in the command window:

```
you have clicked the clickable label!
you have clicked the vbox!
```

- Now click anywhere on the window. You should only see one message output:

```
you have clicked the vbox!
```



Sample Code

```
1  <?php
2  $window = new GtkWindow();
3  $window->set_size_request(200, 100);
4  $window->connect_simple('destroy', array('Gtk','main_quit'));
5
6  $vbox = new GtkVBox();
7  $eventbox = new GtkEventBox; // note 1
8  $eventbox->add($vbox); // note 1
9  $eventbox->connect('button-press-event', 'on_vbox_button_press'); // note 2
10 $window->add($eventbox);
11
12 // create a label
13 $hbox = new GtkHBox();
14 $hbox->pack_start(new GtkLabel("This is a label"), false);
15 $vbox->pack_start($hbox, false);
16
17 // create a clickable label
18 $eventbox = new GtkEventBox;
19 $eventbox->add(new GtkLabel("This is a clickable label"));
20 $eventbox->connect('button-press-event', 'on_button_press'); // note 3
21 $eventbox->modify_bg(Gtk::STATE_NORMAL, GdkColor::parse("#ffff66"));
22 $hbox = new GtkHBox();
23 $hbox->pack_start($eventbox, false);
24 $vbox->pack_start($hbox, false);
25
26
27 $window->show_all();
28 Gtk::main();
29
30 function on_button_press($widget, $event) { // note 4
31    echo "you have clicked the clickable label!\n";
32 }
33
34 function on_vbox_button_press($widget, $event) { // note 5
35    echo "you have clicked the vbox!\n\n";
36 }
37
38 ?>
```

Listing 3.12.php

Explanation

In this example, we capture 'button-press-event' at two places: one at the clickable label, just like the previous example. Another one at the clickable label's parent - the vbox.

1. As explained in Lesson 3.10, vbox is also a widget with no window. To be able to respond to button-press-event, we enclose it in an eventbox, just like the clickable label.
2. Register 'button-press-event' for vbox.
3. Register 'button-press-event' for the clickable label.
4. Signal handler for the clickable label 'button-press-event'.
5. Signal handler for the vbox 'button-press-event'.

Signal Propagate from source upward to its parents

Take a close look at the output. You will see that if you click on the clickable label, the signal handler for the clickable label first respond, followed by that of the vbox.

As explained the overview above, when a 'button-press-event' occurs at the clickable label, the label will first get a chance to respond to this event. Since that callback function did not return TRUE, the signal will continue to propagate upward to its parent, in this case, the vbox. So the vbox's callback function get a chance to attend to the signal too. This may or may not be desirable depending on your application.

Stopping signal propagation

To stop a signal from propagating further, we simply return a `TRUE` at the end of the callback function.

Take a look at the sample code below. Note the addition of return `TRUE` at the end of each callback function.

Run the program again and try clicking on the clickable label. Note that vbox now no longer receives the 'button-press-event' signal.

```
<?php
$window = new GtkWindow();
$window->set_size_request(200, 100);
$window->connect_simple('destroy', array('Gtk','main_quit'));

$vbox = new GtkVBox();
$eventbox = new GtkEventBox;
$eventbox->add($vbox);
$eventbox->connect('button-press-event', 'on_vbox_button_press');
$window->add($eventbox);

// create a label
$hbox = new GtkHBox();
$hbox->pack_start(new GtkLabel("This is a label"), false);
$vbox->pack_start($hbox, false);

// create a clickable label
$eventbox = new GtkEventBox;
$eventbox->add(new GtkLabel("This is a clickable label"));
$eventbox->connect('button-press-event', 'on_button_press');
$eventbox->modify_bg(Gtk::STATE_NORMAL, GdkColor::parse("#ffff66"));
```



```
22 $hbox = new GtkHBox();
23 $hbox->pack_start($eventbox, false);
24 $vbox->pack_start($hbox, false);
25
26
27 $window->show_all();
28 Gtk::main();
29
30 function on_button_press($widget, $event) {
31    echo "you have clicked the clickable label!\n";
32    return true;  // stop further propagation of signal
33 }
34
35 function on_vbox_button_press($widget, $event) {
36    echo "you have clicked the vbox!\n";
37    return true;  // stop further propagation of signal
38 }
39
40 ?>
```

Listing 3.12.1.php

3.13 Handling keypress with key-press-event

Objective

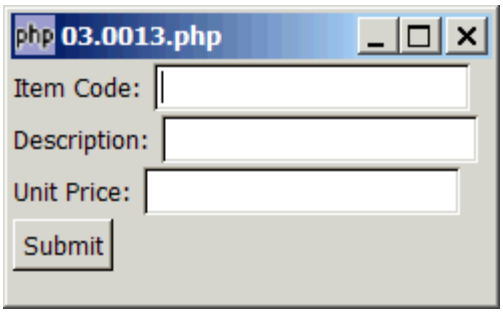
We will now look at how to handle keypress in php-gtk application.

Overview

- Keypress is handled by the signal 'key-press-event'. Every keypress, whether it's a letter, function keys (F1 to F12, PgUp, PgDn, Delete, Return, etc.), or modifier keys (shift, ctrl, alt, etc.) will generate this signal.

Sample Output

- Press F1 in any of the entry fields and see the output message in the command window.
- Now press Tab to move the focus to the button. Press F1 and note the output message.
- Press Ctrl-F12 to exit the application.



Sample Code

Very Important Note: If you have installed php-gtk2 using Gnope Installer on Windows, and if running the sample code below gives you warning that the Symbolic names for keys (e.g. Gdk::KEY_F1) is not defined, you might want to update your php-gtk2 with the latest php-gtk2.dll available at http://www.gnope.org/p/Gnope_PHPGtk2_dll. Simply download the php-gtk2.dll and replace the copy in the folder php-gtk2\ext. The latest compilation has added in the constant definition for the Symbolic names for keys.

```
1  <?php
2  $window = new GtkWindow();
3  $window->connect_simple('destroy',array('Gtk','main_quit'));
4  $window->set_size_request(240, 120);
5  $vbox = new GtkVBox();
6  $window->add($vbox);
7
8  /*// setup a simple menu bar
9  $menubar = new GtkMenuBar();
10 $vbox->pack_start($menubar, false);
11 $menubar->append($top_menu = new GtkMenuItem('_File'));
12 $menu = new GtkMenu();
13 $top_menu->set_submenu($menu);
14 foreach(array('_New', '_Open', '_Close', '_Quit') as $submenu) {
15    $menu->append($menu_item = new GtkMenuItem($submenu));
16 }*/
17
18 $i = 0;
19 foreach(array('Item Code', 'Description', 'Unit Price') as $label) {
20    $hbox = new GtkHBox();
21    $hbox->pack_start(new GtkLabel("$label: "), false);
22    $field_label[$i] = $label;
23    $input[$i] = new GtkEntry();
24    $hbox->pack_start($input[$i], false);
25    $vbox->pack_start($hbox, false);
26    ++$i;
27 }
28
29 $hbox = new GtkHBox();
30 $button = new GtkButton('Submit');
```



```
31 $hbox->pack_start($button, false);
32 $vbox->pack_start($hbox, false);
33
34 $window->connect('key-press-event', 'on_key_press'); // note 1
35 $window->show_all();
36 Gtk::main();
37
38 function on_key_press($widget, $event) { // note 2
39    global $input, $field_label;
40
41    if($event->keyval==Gdk::KEY_F12 && $event->state & Gdk::CONTROL_MASK) { //
   note 3
42       echo "You pressed Ctrl-F12!\n";
43       Gtk::main_quit(); // note 3
44    }
45
46    if ($event->keyval!=Gdk::KEY_F1) return false; // note 4
47    for ($i=0; $i<count($input); ++$i) {
48       if ($input[$i]->is_focus()) { // note 5
49          echo "User pressed F1 in input[$i]: {$field_label[$i]}\n"; // note 5
50          // provide any context sensitive help messages here
51          return true; // note 6
52       }
53    }
54
55    echo "User pressed F1 elsewhere in window\n"; // note 7
56    // provide a generic help message here
57    return true;
58
59 }
60
61 ?>
```

Listing 3.13.php

Explanation

The code above uses Lesson 2.14 as base to display three input fields and a submit button.

What's new here:

1. Register 'key-press-event' on the outermost window.
2. This is the callback function that handles the key-press-event. Note that the two arguments passed to the callback function is very similar to that of the button-press-event.
    - The first argument is the widget that generated the signal, in this case, it's the window.
    - The second argument is the GdkEvent object as mentioned in Lesson 3.11.

3. Here we test for Ctrl-F12. If it's Ctrl-F12, we exit the application.
    - Note that they way we test for control key is exactly the same as that of button-press-event, i.e. using `$event->state & Gdk::CONTROL_MASK`.
    - Note also that php-gtk has pre-defined the symbolic names for all keys. You can view the entire list here. If using the symbolic names gives you warning message, please refer to the note above to update your php-gtk2.dll.
4. If it's not Ctrl-F12, and it's not an F1, we simply return a `FALSE`. As mentioned in the previous example, returning a `false` will cause the signal to continue to propagate. In this case, since we do not have any other signal handler, php-gtk will simply handle the key-press with its default handler.
5. Here we loop through each of the three GtkEntry fields. With the use of the method GtkWidget::is_focus(), we're able to test if the user is currently at any of these fields and display context-sensitive help messages if required.
6. We're done handling the signal. So we return a `TRUE` so that php-gtk will not propagate the signal further.
7. If the program runs till here, it will mean that the user has pressed F1 outside the entry fields.

3.14 Signal propagation for key-press-event

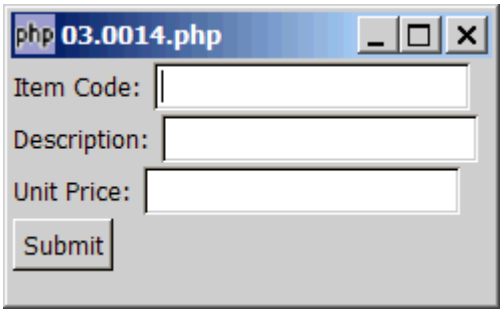
Objective

This example shows how the signal 'key-press-event' gets propagated in php-gtk.

Overview

Unlike button-press-event, 'key-press-event' gets handled by the outmost level first, and travels inward till some widget returns a FALSE.

Sample Output



Sample Code

Very Important Note: If you have installed php-gtk2 using Gnope Installer on Windows, and if running the sample code below gives you warning that the Symbolic names for keys (e.g. Gdk::KEY_F1) is not defined, you might want to update your php-gtk2 with the latest php-gtk2.dll available at

http://www.gnope.org/p/Gnope_PHPGtk2_dll. Simply download the php-gtk2.dll and replace the copy in the folder php-gtk2\ext. The latest compilation has added in the constant definition for the Symbolic names for keys.

```
1  <?php
2  $window = new GtkWindow();
3  $window->connect_simple('destroy',array('Gtk','main_quit'));
4  $window->set_size_request(240, 120);
5  $vbox = new GtkVBox();
6  $window->add($vbox);
7
8  $i = 0;
9  foreach(array('Item Code', 'Description', 'Unit Price') as $label) {
10    $hbox = new GtkHBox();
11    $hbox->pack_start(new GtkLabel("$label: "), false);
12    $field_label[$i] = $label;
13    $input[$i] = new GtkEntry();
14    $input[$i]->connect('key-press-event', 'on_entry_key_press', $i, $label); // note 1
15    $hbox->pack_start($input[$i], false);
16    $vbox->pack_start($hbox, false);
17    ++$i;
18 }
19
20 $hbox = new GtkHBox();
21 $button = new GtkButton('Submit');
22 $hbox->pack_start($button, false);
23 $vbox->pack_start($hbox, false);
24
25 $window->connect('key-press-event', 'on_key_press'); // note 2
26 $window->show_all();
27 Gtk::main();
28
29 function on_key_press($widget, $event) { // note 3
30    if($event->keyval==Gdk::KEY_F12 && $event->state & Gdk::CONTROL_MASK) {
31       echo "You pressed Ctrl-F12!\n";
32       Gtk::main_quit();
33    }
34
35    if ($event->keyval!=Gdk::KEY_F1) return false;
36
37    echo "User pressed F1 elsewhere in window\n";
38    // provide a generic help message here
39    return false; // note 5
40 }
41
42 function on_entry_key_press($widget, $event, $i, $label) { // note 4
43    if ($event->keyval!=Gdk::KEY_F1) return false;
44    echo "User pressed F1 in input[$i]: $label\n";
45    // provide any context sensitive help messages here
46 }
```



```
47
48 ?>
```

Listing 3.14.php

Explanation

In this example, we capture 'key-press-event' at four places: one at the outermost window, just like the previous example. And one each for the three GtkEntry fields.

1. Register 'key-press-event' on the outermost window.
2. Register 'key-press-event' for each of the three GtkEntry field. Note that we pass the index `$i` as well as the label along with the signal.
3. This is the callback function that handles the key-press-event for the outermost window. As in the previous example, if it's a Ctrl-F12, we exit the application. If it's not an F1, we return a FALSE and let the default handler takes care of the keypress.
4. This is the callback function that handles the 'key-press-event' for all the three GtkEntry fields. Since we have passed along the index $i as well as the label, we're able to differentiate which is the entry field that generated the signal.
5. We return a FALSE here to show you how the 'key-press-event' propagates. Try changing this to return TRUE. You will find that the three GtkEntry fields will never receive the signal because the signal gets stopped here.

Propagation of key-press-event

As you can see from the output in the command window, the 'key-press-event' gets handled by the outermost widget first, in this case, the window. If the result is a FALSE, the signal will then propagate to the inner widgets.

So while most signals such as 'button-press-event' travel from source (the widget that emitted the signal) outward, the 'key-press-event' travels in the other direction – inward starting from the outermost level.

Fixing this example

Comparing this example and the previous example, you can see that the code looks much "cleaner" when the GtkEntry field handles its own 'key-press-event'. Imagine you have 100 over widgets in a complicated layout. If there's only one 'key-press-event' handler at the outermost window level, there will be too many if-then-else statements to handle the different keypress demanded by each of the widget.

However for this example, we cannot just put a return TRUE at the line with "note 5". The signal will get stopped at the outermost window, and the GtkEntry fields will never get any signal of F1 key press. On the other hand, we cannot return FALSE either, because the user will then get both the generic help as well as context-sensitive help.

To fix this, we first test if the user is currently in any of the GtkEntry fields with the help of the method GtkWidget::is_focus(). If it is, we simply return a FALSE. Php-gtk will pass it on to the next widget in line, and the signal will be picked up by the callback function which we have set up for the GtkEntry fields.

If the user is not in any of the entry fields, the application can display a generic help message and return a TRUE, so that the signal will not be propagated further.

Below is the revised code:

```
<?php
$window = new GtkWindow();
$window->connect_simple('destroy',array('Gtk','main_quit'));
$window->set_size_request(240, 120);
$vbox = new GtkVBox();
$window->add($vbox);

/*// setup a simple menu bar
$menubar = new GtkMenuBar();
$vbox->pack_start($menubar, false);
$menubar->append($top_menu = new GtkMenuItem('_File'));
$menu = new GtkMenu();
$top_menu->set_submenu($menu);
foreach(array('_New', '_Open', '_Close', '_Quit') as $submenu) {
   $menu->append($menu_item = new GtkMenuItem($submenu));
}*/

$i = 0;
foreach(array('Item Code', 'Description', 'Unit Price') as $label) {
   $hbox = new GtkHBox();
   $hbox->pack_start(new GtkLabel("$label: "), false);
   $field_label[$i] = $label;
   $input[$i] = new GtkEntry();
   $input[$i]->connect('key-press-event', 'on_entry_key_press', $i, $label);
   $hbox->pack_start($input[$i], false);
   $vbox->pack_start($hbox, false);
   ++$i;
}

$hbox = new GtkHBox();
$button = new GtkButton('Submit');
$hbox->pack_start($button, false);
$vbox->pack_start($hbox, false);

$window->connect('key-press-event', 'on_key_press');
$window->show_all();
Gtk::main();

function on_key_press($widget, $event) {
   if($event->keyval==Gdk::KEY_F12 && $event->state & Gdk::CONTROL_MASK) {
      echo "You pressed Ctrl-F12!\n";
      Gtk::main_quit();
   }

```



```
45      if ($event->keyval!=Gdk::KEY_F1) return false;
46
47      global $input;
48      for ($i=0; $i<count($input); ++$i) {
49          if ($input[$i]->is_focus()) {
50              return false;
51          }
52      }
53
54      echo "User pressed F1 elsewhere in window\n";
55      // provide a generic help message here
56      return true;
57  }
58
59  function on_entry_key_press($widget, $event, $i, $label) {
60      if ($event->keyval!=Gdk::KEY_F1) return false;
61      echo "User pressed F1 in input[$i]: $label\n";
62      // provide any context sensitive help messages here
63  }
64
65  ?>
```

Listing 3.14.1.php

3.15 Summary

Here's a quick summary of what we have learned in this chapter:

- Signal handling is what makes a php-gtk application comes to life. It allows you to respond to user's action and interact with the user.
- There are numerous signals continuously monitored by php-gtk. You register with php-gtk the ones you are interested with the connect() statement.
- Each signal is identified by a unique name. This is the string that goes into the first argument of the connect() statement.
- The second argument of the connect() statement is your callback function – the function that gets called the signal you specified is being detected. This is also commonly being referred to as a signal handler.
- You can specify any additional data to be passed along with the signal after the second argument of the connect() statement.
- You can manually generate signals too, e.g., GtkButon::clicked() and GtkTogglebutton::toggled().
- Some widget e.g. GtkLabel, does not have an associated window of its own. They cannot receive events. To receive events, you can wrap these widgets in a GtkEventbox.
- Most signals, e.g. 'button-press-event', propagate from source outward to its parents until one of the widgets return TRUE.
- However, the signal 'key-press-event' travels from the outermost widget inward.

More examples

To see how some of these techniques are being used in practice, you may refer to the following examples in the php-gtk2 Cookbook website:

button-press-event - GtkWidget

- How to put a clickable link in GtkLabel - Part 1?

changed - GtkComboBox

- How to change buttons on the fly based on pulldown menu selections?
- How to display a popup alert for required fields - Part 1?

changed - GtkTreeSelection

- How to display a 2D array in GtkTreeView - Part 5?

clicked - GtkButton

- How to set the background color of GtkButton?
- How to setup and process GtkComboBox?
- How to setup and process GtkComboBoxEntry?

edited - GtkCellRendererText

- How to use GtkCellRendererCombo - Part 2 - process combobox?

- How to use GtkCellRendererCombo - Part 3 - process user selection?

enter-notify-event / leave-notify-event - GtkWidget

- How to put a clickable link in GtkLabel - Part 2?

toggled - GtkToggleButton

- How to display and process grouped radio buttons?

Chapter 4 Object-oriented Framework

PHP5 comes with great object-oriented support and PHP-GTK2 is written entirely using object-oriented architecture.

If you're used to object-oriented programming, and enjoy writing in object-oriented style, you will find yourself very comfortable developing applications with php-gtk2.

What if you have not learned objected-oriented programming before? Don't worry. As you have seen in the many examples in this book, we can use php-gtk2 with just plain old non object-oriented php. You do not need to program in object-oriented style in php in order to use php-gtk2.

You do need, however, to at least understand the object-oriented framework of php-gtk2 – so that you can locate the methods and signals you need.

Important Note: This chapter does not attempt to teach you object-oriented programming. You only need to read Lesson 4.1 if you do not intend to use object-oriented programming with php-gtk2.

However, if you have some background of object-oriented programming, Lesson 4.2 to Lesson 4.6 will show you different ways of using object-oriented programming in php-gtk2.

4.1 The object-oriented widgets

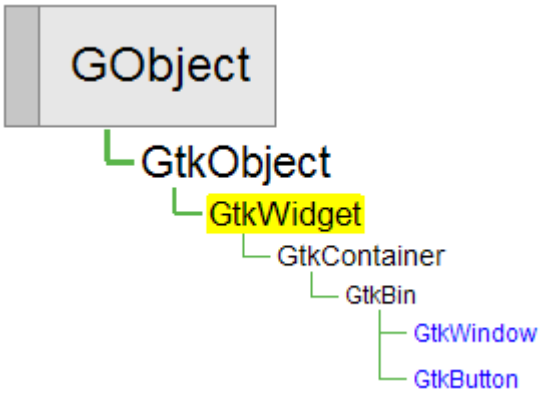
Objective

Let's begin our understanding of the object-oriented nature of php-gtk by taking a look again at Example 2.7. The code is reproduced below.

In this example, we want the button to be exactly of the size 96 x 36.

Overview

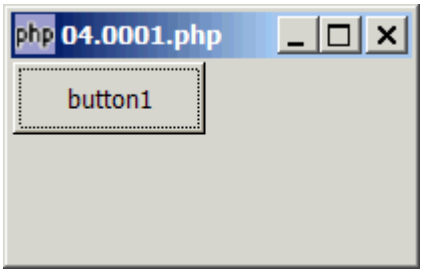
- Both GtkWindow and GtkButton are descendents of GtkWidget as shown in the diagram below.



- If you look at the php-gtk manual for GtkWindow and GtkButton, you do not see any method `set_size_request()`.

- You have to go to GtkWidget to find this method: GtkWidget::set_size_request().
- As GtkWindow and GtkButton are descendents of GtkWidget, they inherit their parent's and ancestor's methods. That's why we can call `set_size_request()` from GtkWindow and GtkButton.

Sample Output



Sample Code

```
1  <?php
2  $window = new GtkWindow();
3  $window->connect_simple('destroy',array('Gtk','main_quit'));
4  $window->set_size_request(200, 100); // note 1
5  $vbox = new GtkVBox();
6  $window->add($vbox);
7
8  $button = new GtkButton('button1');
9  $button->set_size_request(96,36); // note 2
10 $hbox = new GtkHBox();
11 $hbox->pack_start($button, false);
12 $vbox->pack_start($hbox, false);
13
14 $window->show_all();
15 Gtk::main();
16 ?>
```

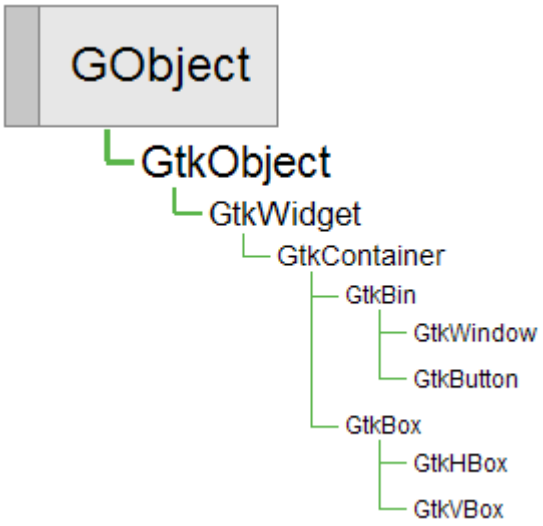
Listing 4.1.php

Explanation

1. Set the size of window using the ancestor's method GtkWidget::set_size_request().
2. 2. Set the size of button also using the ancestor's method GtkWidget::set_size_request().

Other Inherited Methods

Take a look at the example again. We mapped out the parent-child relationship of all the widgets used in this example.

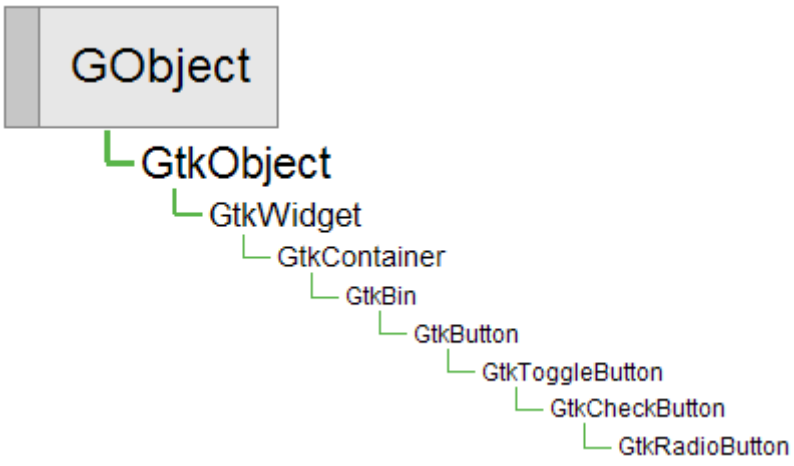


As you can see in the list below, all the methods used in this example are inherited from their ancestors

- $window->connect_simple() is inherited from GObject.
- $window->add() is inherited from GtkContainer.
- $vbox->pack_start() is inherited from GtkBox.
- $window->show_all() is inherited from GtkWidget.

Looking up Inherited Methods, Signals and Properties

This partly explains why learning php-gtk is sometimes frustrating, because a method that you need might be deeply buried somewhere within its ancestors, several levels up. Take a look at the widget GtkRadiobutton.



It has eight ancestors! And guess:

- Which ancestor contains the method get_active()?
- Which ancestor contains the method that allows you to retrieve the label of the radio button, and what is the name of the method?

- Which ancestor can generate the signal that allows you to know when the user hovers the mouse over the radio button? What is the name of the signal?

Through experience you might know the answer right away. Otherwise, there is no easy way out but to look into each of the eight ancestors one by one to look for these methods and signals.

4.2 Objected-oriented programming - Variation 1

Objective

Suppose you enjoy object-oriented programming, and want to develop php-gtk2 applications using object-oriented programming.

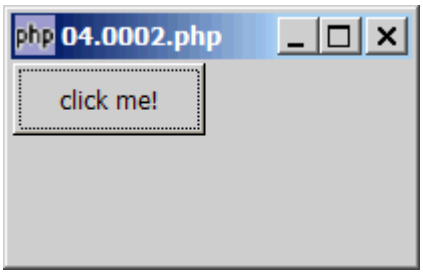
There are many ways of doing this. Let's first look at variation one.

We will use the previous example as base, and rewrite it using object-oriented style.

Overview

- We simply encapsulates the setting of a GtkWindow and display of a button into a class.
- As explained in Lesson 3.7, to connect a signal to a method in a class we use `connect(signal_name, array(class_name, method_name))`.

Sample Output



Sample Code

```
1  <?php
2
3  class App { // note 1
4
5     // the constructor
6     function __construct() {
7        $window = new GtkWindow();
8        $window->connect_simple('destroy',array('Gtk','main_quit'));
9        $window->set_size_request(200, 100);
10       $vbox = new GtkVBox();
11       $window->add($vbox);
12
```



```
13        $button = new GtkButton('click me!');
14        $button->set_size_request(96,36);
15        $hbox = new GtkHBox();
16        $hbox->pack_start($button, false);
17        $vbox->pack_start($hbox, false);
18
19        $button->connect('clicked', array($this, 'on_click')); // note 2
20
21        $window->show_all();
22     }
23
24     function main() {
25        Gtk::main();
26     }
27
28     function on_click($button) {
29        echo "you have clicked me!\n";
30     }
31  }
32
33  $app = new App(); // note 3
34  $app->main(); // note 4
35
36  ?>
```

Listing 4.2.php

Explanation

1. Encapsulate the functionalities into class App.
2. Connect the signal 'clicked' to the method on_click() within the class App.
3. Create a new object of class App.
4. And let's run it!

4.3 Objected-oriented programming - Variation 2

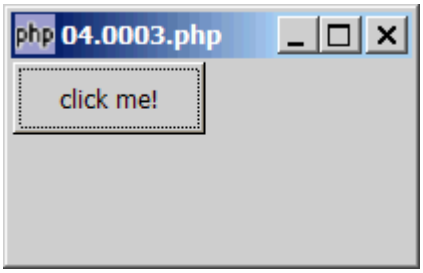
Objective

In Variation 1, we use the standard widget sets in php-gtk2. In this variation, we begin to customize php-gtk widgets to suit our needs.

Overview

- Since php-gtk2 adopts object-oriented architecture, we can customize the Gtk widgets by extending them.
- In this variation, we extend GtkWindow to form a customized GtkWindow.

Sample Output



Sample Code

```
1  <?php
2
3  class MyWindow extends GtkWindow { // note 1
4
5     // the constructor
6     function __construct() {
7        parent::__construct(); // note 2
8        $this->connect_simple('destroy',array('Gtk','main_quit'));
9        $this->set_size_request(200, 100);
10       $vbox = new GtkVBox();
11       $this->add($vbox);
12
13       $button = new GtkButton('click me!');
14       $button->set_size_request(96,36);
15       $hbox = new GtkHBox();
16       $hbox->pack_start($button, false);
17       $vbox->pack_start($hbox, false);
18
19       $button->connect('clicked', array($this, 'on_click'));
20
21       $this->show_all();
22    }
23
24    function on_click($button) {
25       echo "you have clicked me!\n";
26    }
27 }
28
29 $window = new MyWindow(); // note 3
30 Gtk::main(); // note 4
31
32 ?>
```

Listing 4.3.php

Explanation

1. Create a customized GtkWindow.
2. When we extend an existing widget, make sure we first give a call to its parent's constructor: parent::__construct()
3. Create a new instance of the customized GtkWindow.
4. Let's go!

Note

There are no clear advantages or disadvantages between Variation 1 and 2. They are just different ways of modeling the “real world”.

4.4 Objected-oriented programming - Variation 3

Objective

In Variation 2, we simply “extend” an existing widget to add on additional functionality.

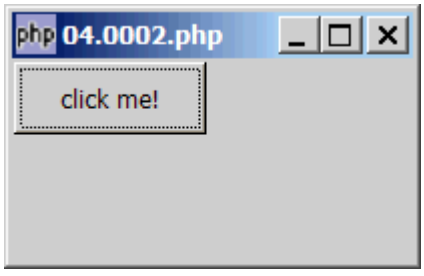
In this variation, we go one step further to encapsulate related properties, methods and signal handlers in a class.

Overview

In this variation,

- We extend GtkWindow to form a customized GtkWindow.
- We also extend GtkButton to form a customized Gtkbutton. The signal handler now resides in this customized button.

Sample Output



Sample Code

```
1  <?php
2
3  class MyWindow extends GtkWindow {
4      function __construct() {
5          parent::__construct();
6          $this->connect_simple('destroy',array('Gtk','main_quit'));
```



```
7         $this->set_size_request(200, 100);
8         $vbox = new GtkVBox();
9         $this->add($vbox);
10
11        $button = new MyButton('click me!', 96, 36); // note 2
12        $hbox = new GtkHBox();
13        $hbox->pack_start($button, false);
14        $vbox->pack_start($hbox, false);
15
16        $this->show_all();
17    }
18 }
19
20 class MyButton extends GtkButton{ // note 1
21    function __construct($label, $width=-1, $height=-1) { // note 3
22        parent::__construct($label);
23        $this->set_size_request($width,$height); // note 3
24        $this->connect('clicked', array($this, 'on_click')); // note 4
25    }
26    function on_click($button) { // note 5
27        echo "you have clicked me!\n";
28    }
29 }
30
31 $window = new MyWindow();
32 Gtk::main();
33
34 ?>
```

Explanation

1. Create a customized GtkButton.
2. Create a new instance of the customized button.
3. Set the size of the customized button. If no width and height is specified, it will use the default size of 96x36.
4. Register the signal 'clicked'.
5. The signal handler for the button is now encapsulated within the class itself.

Note

The highlight of variation 3 is that the signal handler for the button is now encapsulated within the class “MyButton”. As you develop more complex applications, you will find that encapsulation allows you to group related properties, methods and signal handlers neatly in a class. Code will be cleaner, and debugging easier.

4.5 Objected-oriented programming - Variation 4

Objective

Variation 1 simply wraps everything in a class.

Variation 2 extends existing widgets.

Variation 3 encapsulates signal handlers in a class.

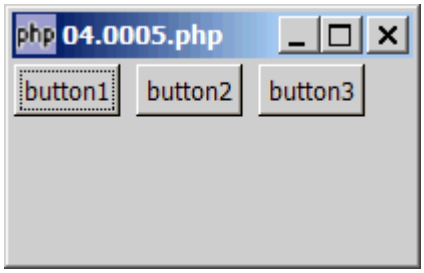
In this variation, we “create” new widgets by customizing existing widgets.

Overview

In this variation,

- We extend GtkWindow to form a customized GtkWindow.
- The standard GtkButton creates only one button. We form a new widget that allows us to hold multiple buttons by extending GtkHBox.

Sample Output



Sample Code

```
1  <?php
2
3  class MyWindow extends GtkWindow {
4
5     function __construct() {
6        parent::__construct();
7        $this->connect_simple('destroy',array('Gtk','main_quit'));
8        $this->set_size_request(200, 100);
9        $vbox = new GtkVBox();
10       $this->add($vbox);
11
12       $buttons = new MyButtons(
13          array('button1', 'button2', 'button3')); // note 2
14       $vbox->pack_start($buttons, false);
15
16       $this->show_all();
17    }
18
19 }
```



```
20
21 class MyButtons extends GtkHBox { // note 1
22
23    var $vbox;
24
25    function __construct($label_array, $width=-1, $height=-1, $spacer_size=4) {
26       parent::__construct();
27       $hbox = new GtkHBox();
28       foreach($label_array as $label) {
29          $button = new GtkButton($label);
30          $button->set_size_request($width,$height);
31          $hbox->pack_start($button, false);
32          $spacer = new GtkHBox();
33          $spacer->set_size_request($spacer_size,-1);
34          $hbox->pack_start($spacer, false);
35          $button->connect('clicked', array($this, 'on_click'), $button);
36       }
37       $this->pack_start($hbox, false);
38    }
39
40    function on_click($button) {
41       $label = $button->get_label();
42       echo "you have clicked $label!\n";
43    }
44 }
45
46 $window = new MyWindow();
47 Gtk::main();
48
49 ?>
```

Listing 4.5.php

Explanation

1. Class definition for a new class `MyButtons`.
2. Create an instance of the class `MyButtons`. The button labels are specified in the form of array.

4.6 Creating your own widgets

Objective

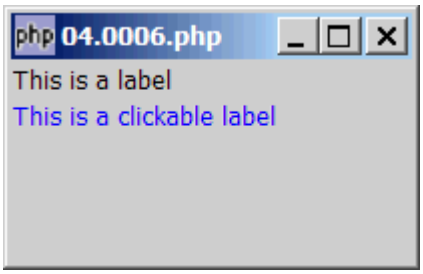
Let's have one more example of creating customized widgets.

Suppose you application uses a lot of clickable labels as described in Lesson 3.10. You can create a new widget called 'ClickableLabel' by extending GtkEventBox.

Overview

- First create a GtkEventBox.
- Create a GtkLabel and stuff it inside the eventbox.

Sample Output



Sample Code

```
1  <?php
2
3  class MyWindow extends GtkWindow {
4
5     function __construct() {
6        parent::__construct();
7        $this->connect_simple('destroy',array('Gtk','main_quit'));
8        $this->set_size_request(200, 100);
9        $vbox = new GtkVBox();
10       $this->add($vbox);
11
12
13       // create a label
14       $hbox = new GtkHBox();
15       $hbox->pack_start(new GtkLabel("This is a label"), false);
16       $vbox->pack_start($hbox, false);
17
18       // create a clickable label
19       $clickable_label = new ClickableLabel('This is a clickable label'); // note 2
20       $clickable_label->connect('button-press-event',
21          array($this, 'on_button_press')); // note 3
22       $hbox = new GtkHBox();
23       $hbox->pack_start($clickable_label, false);
24       $vbox->pack_start($hbox, false);
25
26       $this->show_all();
27    }
28
29    function on_button_press($widget, $event) { // note 3
30       echo "you have clicked the clickable label!\n";
31    }
```



```
32
33 }
34
35 class ClickableLabel extends GtkEventBox {// note 1
36
37    function __construct($label_str, $fg='#0000ff') {
38       parent::__construct();
39       $label = new GtkLabel($label_str);
40       $this->add($label);
41       $label->modify_fg(Gtk::STATE_NORMAL, GdkColor::parse($fg));
42    }
43
44 }
45
46 $window = new MyWindow();
47 Gtk::main();
48
49 ?>
```

Listing 4.6.php

Explanation

1. Class definition for the new widget `ClickableLabel`.
2. Create an instance of the class `ClickableLabel`.
3. Note that this time round the signal handler resides in the class GtkWindow and not within the `ClickableLabel` class definition.

4.7 Summary

Here's a quick summary of what we have learned in this chapter:

- PHP-GTK2 is written entirely using the object-oriented architecture.
- Child widgets inherit the properties, methods and signals of its ancestors.
- To find a method or signal, sometimes you need to look into each of the ancestors.
- It's not necessary to use object-oriented programming to use php-gtk2. You can still write full-fledge php-gtk2 applications using plain old non-object-oriented php.
- The most important thing is that you understand the object-oriented framework of php-gtk2 – so that you can locate the methods and signals you need,
- Of course, you will find php-gtk2 more fun to by taking advantage of the object-oriented feature. For example, you can extend existing widgets to form new widgets.
- With objected-oriented programming, you can also encapsulate business logic and relationships into classes.
- There are many ways of using object-oriented programming in php-gtk2. This chapter shows four variations to a simple script that displays a GtkButton in a GtkWindow.

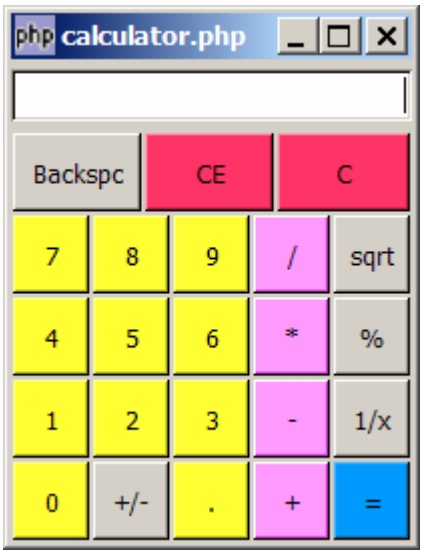
More examples

To see how some of these techniques are being used in practice, you may refer to the following examples in the php-gtk2 Cookbook website:

- Calculator
- Application Template 01 - multiple modules in multiple window
- Application Template 02 - Multiple Modules in Multiple Windows

Chapter 5 Putting It Altogether

In this chapter, we will put together all that we have learned in the previous chapters to develop a simple calculator, similar to the windows calculator as shown below:



We will develop this in five steps:

1. Layout the widgets
2. Setup signal handlers
3. Add core business logic
4. Add validation checks
5. Resize of window

Note

The php-gtk2 Cookbook website also has a sample application of calculator. The output looks the same. But the codes are very different.

If you have understood the techniques described so far, you will find that the approach used in this book is cleaner and more elegant.

5.1 Layout the widgets

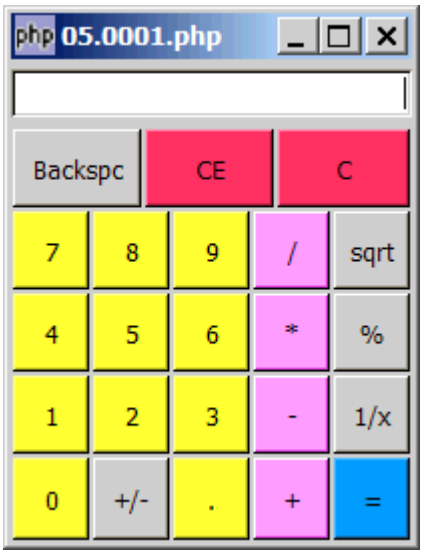
Objective

In developing a php-gtk2 applications, the first step we usually do is to layout the widgets first.

Overview

- Extends GtkWindow to form our customized window. Almost all the widget layout codes will be encapsulated in this class. A lot of the codes are only used once during the initial setup. So we can hide these codes nicely inside the class.
- Extends GtkButton for the calculator buttons. The setting of button size and its background color are all encapsulated within this class.

Sample Output



Sample Code

```
1  <?php
2
3  class Calculator{
4
5     // constructor
6     function __construct() {
7     }
8
9     function main() {
10       $this->window = new CalculatorWindow();
11       $this->window->show_all(); // display the calculator
```



```
12        Gtk::main(); // and let's go!
13    }
14 }
15
16 class CalculatorWindow extends GtkWindow { // note 1
17    function __construct() {
18       parent::__construct();
19       $this->set_size_request(200, 240);
20       $this->connect_simple('destroy', array('Gtk','main_quit'));
21       $this->connect('key-press-event', array(&$this, "on_keypress"));
22       $this->layout_widgets();
23    }
24
25    function layout_widgets() {
26       // setup a vbox to hold display and buttons
27       $vbox = new GtkVBox();
28       $this->add($vbox);
29
30       // setup display
31       $this->display = new GtkEntry(); // note 2
32       $this->display->set_alignment(1.0); // right-justified
33       $this->display->set_editable(false); // for display only
34       $vbox->pack_start($this->display);
35
36       // the button labels
37       $button_label = array( // note 3
38       array('Backspc', 'CE', 'C'),
39       array('7', '8', '9', '/', 'sqrt'),
40       array('4', '5', '6', '*', '%'),
41       array('1', '2', '3', '-', '1/x'),
42       array('0', '+/-', '.', '+', '=')
43       );
44
45       // setup the buttons
46       for ($j=0; $j<5; ++$j) { // note 4
47          $hbox = new GtkHBox();
48          $vbox->pack_start($hbox); // use a hbox to hold each row of buttons
49          for ($i=0; $i<5; ++$i) {
50             if ($j==0 && $i>2) continue; // first row contains only 3 buttons
51             $button = new CalculatorButton($button_label[$j][$i]);
52             $hbox->pack_start($button);
53          }
54       }
55    }
56 }
57
58 class CalculatorButton extends GtkButton { // note 5
59    function __construct($label, $width=-1, $height=-1) {
60       parent::__construct($label);
61       $this->set_size_request(40, 32); // note 6
```



```
62
63          if (preg_match("/^([0-9]|\.)$/", $label)) // note 7
64          $this->modify_bg(Gtk::STATE_NORMAL, GdkColor::parse('#FFFF33'));
65          if (preg_match("/^(\+|-|\*|\/){1}$/", $label))
66          $this->modify_bg(Gtk::STATE_NORMAL, GdkColor::parse('#ff99ff'));
67          if ($label=='C' || $label=='CE')
68          $this->modify_bg(Gtk::STATE_NORMAL, GdkColor::parse('#FF3366'));
69          if ($label=='=')
70          $this->modify_bg(Gtk::STATE_NORMAL, GdkColor::parse('#0099FF'));
71
72      }
73  }
74
75  $cal = new Calculator(); // cerate a new calculator
76  $cal->main(); // let's run it!
77
78  ?>
```

Listing 5.1.php

Explanation

1. Extends GtkWindow to form our customized window.
2. Set up the display.
3. The array that stores the calculation buttons definition.
4. Layout the calculator buttons.
5. Extends GtkButton.
6. Set the size of the buttons.
7. Set the background color of the buttons.

5.2 Set up signal handlers

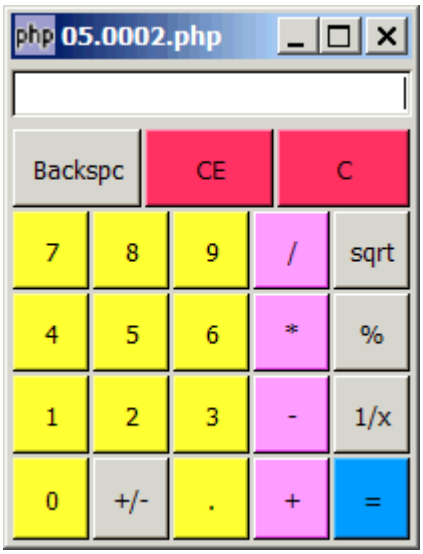
Objective

After we have laid out the widgets, the next step is to set up the signal handlers to respond to user's inputs.

Overview

- Extends GtkWindow to form our customized window. Almost all the widget layout codes will be encapsulated in this class. A lot of the codes are only used once during the initial setup. So we can hide these codes nicely inside the class.
- Extends GtkButton for the calculator buttons. The setting of button size and its background color are all encapsulated within this class.

Sample Output



Sample Code

```
1  <?php
2
3  class Calculator{
4
5      private $operator = '';
6      private $stack = array();
7      private $is_entering_number = 0;
8      private $window;
9
10     // constructor
11     function __construct() {
```



```
12      }
13
14      function main() {
15          $this->window = new CalculatorWindow($this); // create a new calculator
16          $this->window->show_all(); // display the calculator
17          Gtk::main(); // and let's go!
18      }
19  }
20
21  class CalculatorWindow extends GtkWindow {
22      private $calculator;
23      private $display;
24      private $vbox;
25
26      function __construct($calculator) {
27          parent::__construct();
28          $this->calculator = $calculator;
29          $this->set_size_request(200, 240);
30          $this->connect_simple('destroy', array('Gtk','main_quit'));
31          $this->connect('key-press-event', array(&$this, "on_keypress")); // note 1
32          $this->layout_widgets();
33      }
34
35      private function layout_widgets() {
36          // setup a vbox to hold display and buttons
37          $vbox = new GtkVBox();
38          $this->add($vbox);
39
40          // setup display
41          $this->display = new GtkEntry();
42          $this->display->set_alignment(1.0); // right-justified
43          $this->display->set_editable(false); // for display only
44          $vbox->pack_start($this->display);
45
46          // the button labels
47          $button_label = array(
48          array('Backspc', 'CE', 'C'),
49          array('7', '8', '9', '/', 'sqrt'),
50          array('4', '5', '6', '*', '%'),
51          array('1', '2', '3', '-', '1/x'),
52          array('0', '+/-', '.', '+', '=')
53          );
54
55          // setup the buttons
56          for ($j=0; $j<5; ++$j) {
57              $hbox = new GtkHBox();
58              $vbox->pack_start($hbox); // use a hbox to hold each row of buttons
59              for ($i=0; $i<5; ++$i) {
60                  if ($j==0 && $i>2) continue; // first row contains only 3 buttons
61                  $button = new CalculatorButton($button_label[$j][$i]);
```



```
62              $hbox->pack_start($button);
63            }
64        }
65    }
66
67    public function on_keypress($widget, $event) { // note 2
68        $keyval = $event->keyval;
69        $value = chr($keyval);
70        echo "keypress = $keyval ($value)\n";
71    }
72
73 }
74
75 class CalculatorButton extends GtkButton {
76    function __construct($label, $width=-1, $height=-1) {
77        parent::__construct($label);
78        $this->set_size_request(40, 32);
79        $this->connect('clicked', array($this, 'on_button')); // note 3
80
81        if (preg_match("/^([0-9]|\.)$/", $label))
82        $this->modify_bg(Gtk::STATE_NORMAL, GdkColor::parse('#FFFF33'));
83        if (preg_match("/^(\+|-|\*|\/){1}$/", $label))
84        $this->modify_bg(Gtk::STATE_NORMAL, GdkColor::parse('#ff99ff'));
85        if ($label=='C' || $label=='CE')
86        $this->modify_bg(Gtk::STATE_NORMAL, GdkColor::parse('#FF3366'));
87        if ($label=='=')
88        $this->modify_bg(Gtk::STATE_NORMAL, GdkColor::parse('#0099FF'));
89    }
90
91    function on_button($button) { // note 4
92        $value = $button->child->get_text();
93        echo "button_click: $value\n";
94    }
95 }
96
97 $cal = new Calculator();
98 $cal->main();
99
100 ?>
```

Listing 5.2.php

Explanation

1. Extends GtkWindow to form our customized window.
2. Set up the display.
3. The array that stores the calculation buttons definition.
4. Layout the calculator buttons.
5. Extends GtkButton.

6. Set the size of the buttons.
7. Set the background color of the buttons.

Note

Note that the signal handlers now only echo the user's keypress or button clicks. They do not perform any calculations yet. We will put in the business logic in the next lesson.

5.3 Add in core business logic

Objective

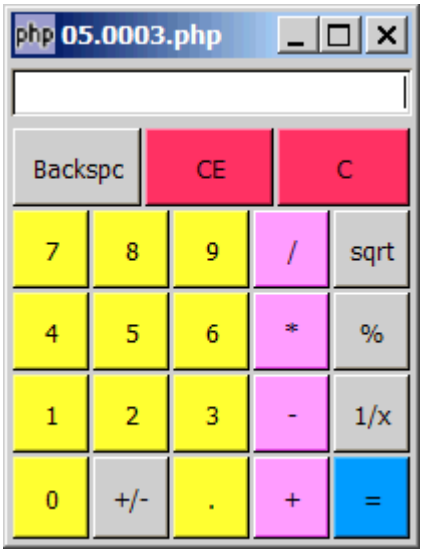
In this step, we will put in all the core business logic. At the end of this step, you will have a fully functional calculator.

By "core", we mean it will function, provided the user don't key in anything funny, such as "1.2.3 * + 456". We will put in the validation checks in Step 4.

Overview

We usually implement a calculator using a *stack*. A stack is just like a paper tray. The first one that goes in stays at the bottom, and you keep piling documents up. It's a list-in-first-out (LIFO) list. Compare this with a queue – which is a first-in-first-out (FIFO) list.

Sample Output



Sample Code

```
1  <?php
2
3  class Calculator{
4
5      private $operator = '';
6      private $stack = array();
7      private $is_entering_number = 0;
8      private $window;
9
10     // constructor
```



```
11      function __construct() {
12      }
13
14      function main() {
15          $this->window = new CalculatorWindow($this); // create a new calculator
16          $this->window->show_all(); // display the calculator
17          Gtk::main(); // and let's go!
18      }
19
20      // process input
21      function process_input($value) { // note 1
22          if (preg_match("/^([0-9]|\.)$/", $value)) { // note 2
23              $number = array_pop($this->stack);
24              if (preg_match("/^([0-9]|\.)+$/", $number)) {
25                  $number .= $value;
26              } else {
27                  if ($number!='') array_push($this->stack, $number);
28                  $number = $value;
29              }
30              array_push($this->stack, $number);
31              $this->window->set_display($number);
32
33          } elseif (preg_match("/^(\+|-|\*|\/|=){1}$/", $value)) { // note 3
34              if (count($this->stack)<3) {
35                  array_push($this->stack, $value);
36              } else {
37                  $number2 = array_pop($this->stack);
38                  $operator = array_pop($this->stack);
39                  $number1 = array_pop($this->stack);
40                  switch ($operator) {
41                      case '+': $result = $number1 + $number2; break;
42                      case '-': $result = $number1 - $number2; break;
43                      case '*': $result = $number1 * $number2; break;
44                      case '/': $result = $number1 / $number2; break;
45                  }
46                  $this->window->set_display($result);
47                  array_push($this->stack, $result);
48                  if ($value!='=') array_push($this->stack, $value);
49              }
50
51          } elseif ($value=='1/x' || $value=='sqrt' || $value=='%' || $value=='+/-') { //
    note 4
52              if (count($this->stack)>=1) {
53                  $number = array_pop($this->stack);
54                  switch ($value) {
55                      case '1/x': $result = 1/$number; break;
56                      case 'sqrt': $result = sqrt($number); break;
57                      case '%': $result = $number / 100; break;
58                      case '+/-': $result = -$number; break;
59                  }
```



```
60              $this->window->set_display($result);
61              array_push($this->stack, $result);
62           }
63
64        } elseif ($value=='C') { // note 5
65           $this->stack=array();
66           $this->window->set_display('0');
67
68        } elseif ($value=='CE') { // note 6
69           if (preg_match("/^([0-9]|\.)+$/", end($this->stack))) {
70              $number = array_pop($this->stack);
71              $this->window->set_display('0');
72           }
73
74        } elseif ($value=='BackSpc') { // note 7
75           if (preg_match("/^([0-9]|\.)+$/", end($this->stack))) {
76              $number = array_pop($this->stack);
77              $number = substr($number, 0, strlen($number)-1);
78              $this->window->set_display($number);
79              array_push($this->stack, $number);
80           }
81        }
82        $this->show_stack(); // show current stack
83     }
84
85     // prints the current stack content
86     function show_stack() { // note 8
87        echo "current stack content:\n";
88        for ($i=count($this->stack)-1; $i>=0; --$i) {
89           echo "stack[$i] = {$this->stack[$i]}\n";
90        }
91        echo "\n";
92     }
93  }
94
95  class CalculatorWindow extends GtkWindow {
96     private $calculator;
97     private $display;
98     private $vbox;
99
100    function __construct($calculator) {
101       parent::__construct();
102       $this->calculator = $calculator;
103       $this->set_size_request(200, 240);
104       $this->connect_simple('destroy', array('Gtk','main_quit'));
105       $this->connect('key-press-event', array(&$this, "on_keypress"));
106       $this->layout_widgets();
107    }
108
109    private function layout_widgets() {
```



```
110       // setup a vbox to hold display and buttons
111       $vbox = new GtkVBox();
112       $this->add($vbox);
113
114       // setup display
115       $this->display = new GtkEntry();
116       $this->display->set_alignment(1.0); // right-justified
117       $this->display->set_editable(false); // for display only
118       $vbox->pack_start($this->display);
119
120       // the button labels
121       $button_label = array(
122       array('Backspc', 'CE', 'C'),
123       array('7', '8', '9', '/', 'sqrt'),
124       array('4', '5', '6', '*', '%'),
125       array('1', '2', '3', '-', '1/x'),
126       array('0', '+/-', '.', '+', '=')
127       );
128
129       // setup the buttons
130       for ($j=0; $j<5; ++$j) {
131          $hbox = new GtkHBox();
132          $vbox->pack_start($hbox); // use a hbox to hold each row of buttons
133          for ($i=0; $i<5; ++$i) {
134             if ($j==0 && $i>2) continue; // first row contains only 3 buttons
135             $button = new CalculatorButton($button_label[$j][$i], $this->calculator);
136             $hbox->pack_start($button);
137          }
138       }
139    }
140
141    public function on_keypress($widget, $event) {
142       $keyval = $event->keyval;
143       $value = chr($keyval);
144       if (preg_match("/[0-9\+\-\*\/\.=]/", $value) || $keyval==Gdk::KEY_Return ||
    $keyval==Gdk::KEY_BackSpace) {
145          if ($keyval==Gdk::KEY_Return) $value='=';
146          if ($keyval==Gdk::KEY_BackSpace) $value='BackSpc';
147          $this->calculator->process_input($value);
148          $this->display->set_position(-1); // move the cursor to the end of display
149       }
150    }
151
152    public function set_display($value) {
153       print "value = $value\n";
154       $this->display->set_text($value);
155    }
156 }
157
158 class CalculatorButton extends GtkButton {
```



```
159
160     private $calculator;
161
162     function __construct($label, $calculator) {
163         parent::__construct($label);
164         $this->calculator = $calculator;
165         $this->set_size_request(40, 32);
166         $this->connect('clicked', array($this, 'on_button'));
167
168         if (preg_match("/^([0-9]|\.)$/", $label))
169             $this->modify_bg(Gtk::STATE_NORMAL, GdkColor::parse('#FFFF33'));
170         if (preg_match("/^(\+|-|\*|\/){1}$/", $label))
171             $this->modify_bg(Gtk::STATE_NORMAL, GdkColor::parse('#ff99ff'));
172         if ($label=='C' || $label=='CE')
173             $this->modify_bg(Gtk::STATE_NORMAL, GdkColor::parse('#FF3366'));
174         if ($label=='=')
175             $this->modify_bg(Gtk::STATE_NORMAL, GdkColor::parse('#0099FF'));
176     }
177
178     function on_button($button) {
179         $value = $button->child->get_text();
180         echo "button_click: $value\n";
181         $this->calculator->process_input($value);
182     }
183 }
184
185 $cal = new Calculator();
186 $cal->main();
187
188 ?>
```

Listing 5.3.php

Explanation

1. Function to perform calculations.
2. Process digits.
3. Process binary operations.
4. Process unary operators.
5. Process button ‘C’.
6. Process button ‘CE’.
7. Process button ‘BackSpc’.
8. The function show_stack() is just for debugging purpose. It gives a visual printout of the content of the stack as shown above. Note again that the first one that goes in (1234) stays at the bottom, and the latest that goes in (56) is at the top.

5.4 Add validation checks

Objective

In this step, we will try to "foolproof" our calculator by putting in some validation checks .

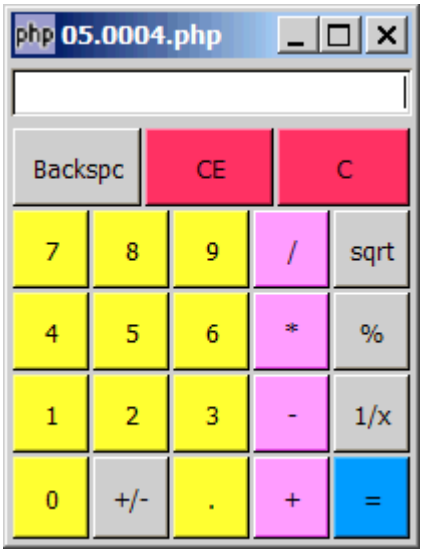
Overview

Below are some of the validation checks that we have put in:

- Don't allow two decimal points e.g. 1.2.3
- + - * / - will only be added to stack if top of stack is a number
- If there is only one number on stack, and the user press '=', the '=' should be go to stack.

There are many more other validation checks that need to put in place if you really want to make this calculator "foolproof". We won't go into all since the focus here is php-gtk2 as oppose to calculator.

Sample Output



Sample Code

```
1  <?php
2
3  class Calculator{
4
5      private $operator = '';
6      private $stack = array();
```



```
7     private $is_entering_number = 0;
8     private $window;
9
10    // constructor
11    function __construct() {
12    }
13
14    function main() {
15       $this->window = new CalculatorWindow($this); // create a new calculator
16       $this->window->show_all(); // display the calculator
17       Gtk::main(); // and let's go!
18    }
19
20    // process input
21    function process_input($value) {
22       if (preg_match("/^([0-9]|\.)$/", $value)) {
23          $number = array_pop($this->stack);
24          if (preg_match("/^([0-9]|\.)+$/", $number)) {
25             if ($value=='.') {
26                if (strpos($number,'.')===false) $number .= $value; // note 1
27             } else {
28                $number .= $value;
29             }
30          } else {
31             if ($number!='') array_push($this->stack, $number);
32             $number = $value;
33          }
34          array_push($this->stack, $number);
35          $this->window->set_display($number);
36
37       } elseif (preg_match("/^(\+|-|\*|\/|=){1}$/", $value)) { // perform binary
operations
38
39          if (count($this->stack)<3) {
40
41             if (preg_match("/^([0-9]|\.)+$/", end($this->stack))) // note 2
42             if ($value!='=') // note 3
43             array_push($this->stack, $value);
44
45          } else {
46
47             $number2 = array_pop($this->stack);
48             $operator = array_pop($this->stack);
49             $number1 = array_pop($this->stack);
50             switch ($operator) {
51                case '+': $result = $number1 + $number2; break;
52                case '-': $result = $number1 - $number2; break;
53                case '*': $result = $number1 * $number2; break;
54                case '/': $result = $number1 / $number2; break;
55             }
```



```
56              $this->window->set_display($result);
57              array_push($this->stack, $result);
58              if ($value!='=') array_push($this->stack, $value);
59           }
60
61           //process unary operators
62        } elseif ($value=='1/x' || $value=='sqrt' || $value=='%' || $value=='+/-') {
63
64           if (count($this->stack)>=1) {
65              $number = array_pop($this->stack);
66              switch ($value) {
67                 case '1/x': $result = 1/$number; break;
68                 case 'sqrt': $result = sqrt($number); break;
69                 case '%': $result = $number / 100; break;
70                 case '+/-': $result = -$number; break;
71              }
72              $this->window->set_display($number);
73              array_push($this->stack, $result);
74
75           }
76
77        } elseif ($value=='C') {
78           $this->stack=array();
79           $this->window->set_display('0');
80
81        } elseif ($value=='CE') {
82           if (preg_match("/^([0-9]|\.)+$/", end($this->stack))) {
83              $number = array_pop($this->stack);
84              $this->window->set_display('0');
85           }
86
87        } elseif ($value=='BackSpc') {
88
89           if (preg_match("/^([0-9]|\.)+$/", end($this->stack))) {
90              $number = array_pop($this->stack);
91              $number = substr($number, 0, strlen($number)-1);
92              $this->window->set_display($number);
93              array_push($this->stack, $number);
94           }
95
96        }
97        $this->show_stack(); // show current stack
98     }
99
100    // prints the current stack content
101    function show_stack() {
102       echo "current stack content:\n";
103       for ($i=count($this->stack)-1; $i>=0; --$i) {
104          echo "stack[$i] = {$this->stack[$i]}\n";
105       }
```



```
106         echo "\n";
107     }
108 }
109
110 class CalculatorWindow extends GtkWindow {
111     private $calculator;
112     private $display;
113     private $vbox;
114
115     function __construct($calculator) {
116         parent::__construct();
117         $this->calculator = $calculator;
118         $this->set_size_request(200, 240);
119         $this->connect_simple('destroy', array('Gtk','main_quit'));
120         $this->connect('key-press-event', array(&$this, "on_keypress"));
121         $this->layout_widgets();
122     }
123
124     private function layout_widgets() {
125         // setup a vbox to hold display and buttons
126         $vbox = new GtkVBox();
127         $this->add($vbox);
128
129         // setup display
130         $this->display = new GtkEntry();
131         $this->display->set_alignment(1.0); // right-justified
132         $this->display->set_editable(false); // for display only
133         $vbox->pack_start($this->display);
134
135         // the button labels
136         $button_label = array(
137         array('Backspc', 'CE', 'C'),
138         array('7', '8', '9', '/', 'sqrt'),
139         array('4', '5', '6', '*', '%'),
140         array('1', '2', '3', '-', '1/x'),
141         array('0', '+/-', '.', '+', '=')
142         );
143
144         // setup the buttons
145         for ($j=0; $j<5; ++$j) {
146             $hbox = new GtkHBox();
147             $vbox->pack_start($hbox); // use a hbox to hold each row of buttons
148             for ($i=0; $i<5; ++$i) {
149                 if ($j==0 && $i>2) continue; // first row contains only 3 buttons
150                 $button = new CalculatorButton($button_label[$j][$i], $this->calculator);
151                 $hbox->pack_start($button);
152             }
153         }
154     }
155
```



```
156    public function on_keypress($widget, $event) {
157
158        $keyval = $event->keyval;
159        $value = chr($keyval); // get the ASCII equivalent
160        if (preg_match("/[0-9\+\-\*\/\.=]/", $value) || $keyval==Gdk::KEY_Return ||
   $keyval==Gdk::KEY_BackSpace) {
161
162            if ($keyval==Gdk::KEY_Return) $value='=';
163            if ($keyval==Gdk::KEY_BackSpace) $value='BackSpc';
164            $this->calculator->process_input($value);
165            $this->display->set_position(-1); // move the cursor to the end of display
166
167        }
168    }
169
170    public function set_display($value) {
171        print "value = $value\n";
172        $this->display->set_text($value);
173    }
174 }
175
176 class CalculatorButton extends GtkButton {
177
178    private $calculator;
179
180    function __construct($label, $calculator) {
181        parent::__construct($label);
182        $this->calculator = $calculator;
183        $this->set_size_request(40, 32);
184        $this->connect('clicked', array($this, 'on_button'));
185
186        if (preg_match("/^([0-9]|\.)$/", $label))
187            $this->modify_bg(Gtk::STATE_NORMAL, GdkColor::parse('#FFFF33'));
188        if (preg_match("/^(\+|-|\*|\/){1}$/", $label))
189            $this->modify_bg(Gtk::STATE_NORMAL, GdkColor::parse('#ff99ff'));
190        if ($label=='C' || $label=='CE')
191            $this->modify_bg(Gtk::STATE_NORMAL, GdkColor::parse('#FF3366'));
192        if ($label=='=')
193            $this->modify_bg(Gtk::STATE_NORMAL, GdkColor::parse('#0099FF'));
194    }
195
196    function on_button($button) {
197
198        $value = $button->child->get_text();
199        echo "button_click: $value\n";
200        $this->calculator->process_input($value);
201
202    }
203 }
204
```



```
205 $cal = new Calculator();
206 $cal->main();
207
208 ?>
```

Listing 5.4.php

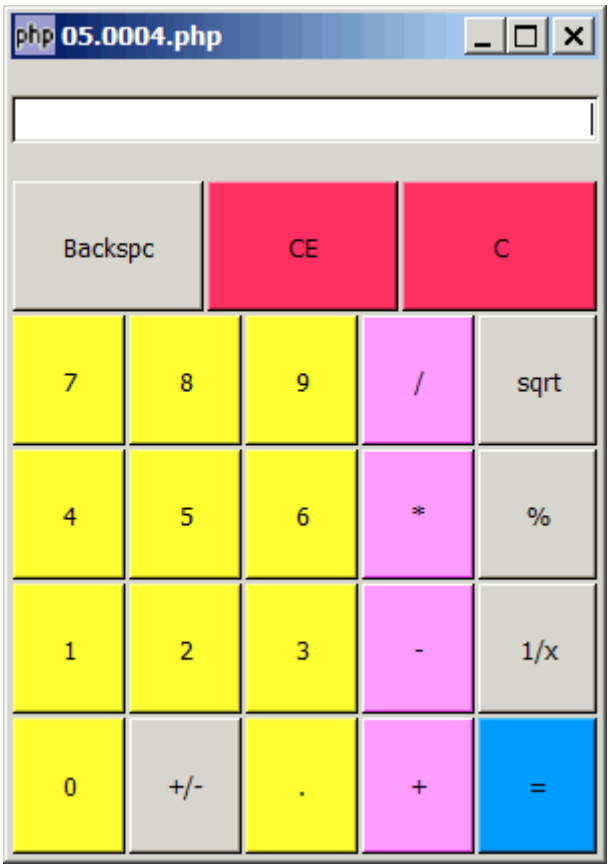
Explanation

1. don't allow two decimal points e.g. 1.2.3
2. + - * / - will only be added to stack if top of stack is a number.
3. if there is only one number on stack, and the user press '=', the '=' should be go to stack.

5.5 Resize of window

Objective

If you find the calculator buttons too small, no worries! Just resize the window to get this:



This is one “cool” feature of php-gtk2 as oppose to tools like Visual Basic. If you have code it the right way, you will find that php-gtk2 will automatically rearrange and resize the widgets when you resize the windows – without the need of you to do any programming.

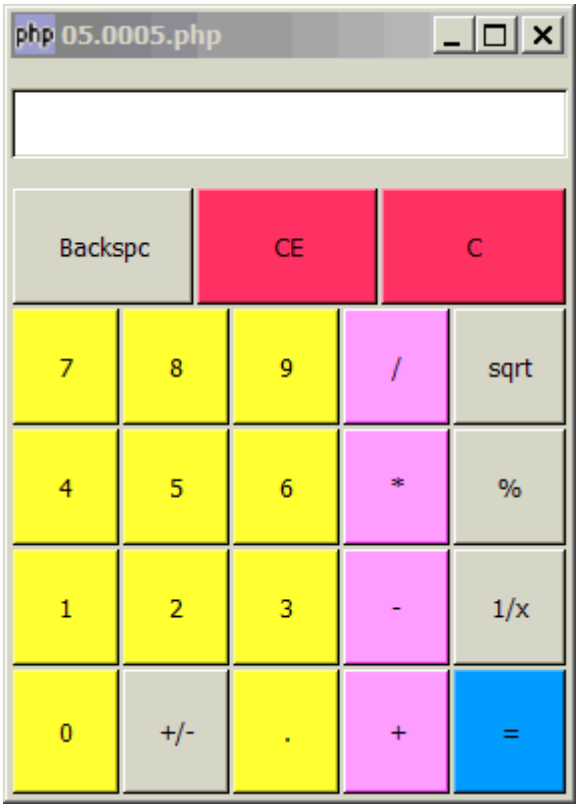
Of course, we cannot expect php-gtk2 to be perfect every time. For example, you will observe that when you enlarge the calculator, the buttons get bigger, but not the display.

In this step, we will adjust the size of the display when the user changes the window size.

Overview

- First we register the signal ‘event’ on GtkWindow to be notified when the user changes the window size.
- To change the size of the display, we use GtkWidget::modify_font().

Sample Output



Sample Code

```
1  <?php
2
3  class Calculator{
4
5     private $operator = '';
6     private $stack = array();
7     private $is_entering_number = 0;
8     private $window;
9
10    // constructor
11    function __construct() {
12    }
13
14    function main() {
15       $this->window = new CalculatorWindow(); // create a new calculator
16       $this->window->show_all(); // display the calculator
17       Gtk::main(); // and let's go!
18    }
19
```



```
20     // process input
21     function process_input($value) {
22        if (preg_match("/^([0-9]|\.)$/", $value)) {
23           $number = array_pop($this->stack);
24           if (preg_match("/^([0-9]|\.)+$/", $number)) {
25              if ($value=='.') {
26                 if (strpos($number,'.')===false) $number .= $value;
27              } else {
28                 $number .= $value;
29              }
30           } else {
31              if ($number!='') array_push($this->stack, $number);
32              $number = $value;
33           }
34           array_push($this->stack, $number);
35           $this->window->set_display($number);
36
37        } elseif (preg_match("/^(\+|-|\*|\/|=){1}$/", $value)) { // perform binary
   operations
38           if (count($this->stack)<3) {
39              if (preg_match("/^([0-9]|\.)+$/", end($this->stack)))
40                 if ($value!='=')
41                    array_push($this->stack, $value);
42           } else {
43              $number2 = array_pop($this->stack);
44              $operator = array_pop($this->stack);
45              $number1 = array_pop($this->stack);
46              switch ($operator) {
47                 case '+': $result = $number1 + $number2; break;
48                 case '-': $result = $number1 - $number2; break;
49                 case '*': $result = $number1 * $number2; break;
50                 case '/': $result = $number1 / $number2; break;
51              }
52
53              $this->window->set_display($result);
54              array_push($this->stack, $result);
55              if ($value!='=') array_push($this->stack, $value);
56           }
57
58        //process unary operators
59        } elseif ($value=='1/x' || $value=='sqrt' || $value=='%' || $value=='+/-') {
60
61           if (count($this->stack)>=1) {
62              $number = array_pop($this->stack);
63              switch ($value) {
64                 case '1/x': $result = 1/$number; break;
65                 case 'sqrt': $result = sqrt($number); break;
66                 case '%': $result = $number / 100; break;
67                 case '+/-': $result = -$number; break;
68              }
```



```
69              $this->window->set_display($number);
70              array_push($this->stack, $result);
71           }
72        } elseif ($value=='C') {
73           $this->stack=array();
74           $this->window->set_display('0');
75
76        } elseif ($value=='CE') {
77           if (preg_match("/^([0-9]|\.)+$/", end($this->stack))) {
78              $number = array_pop($this->stack);
79              $this->window->set_display('0');
80           }
81        } elseif ($value=='BackSpc') {
82           if (preg_match("/^([0-9]|\.)+$/", end($this->stack))) {
83              $number = array_pop($this->stack);
84              $number = substr($number, 0, strlen($number)-1);
85              $this->window->set_display($number);
86              array_push($this->stack, $number);
87           }
88        }
89        $this->show_stack(); // show current stack
90     }
91
92     // prints the current stack content
93     function show_stack() {
94        echo "current stack content:\n";
95        for ($i=count($this->stack)-1; $i>=0; --$i) {
96           echo "stack[$i] = {$this->stack[$i]}\n";
97        }
98        echo "\n";
99     }
100 }
101
102 class CalculatorWindow extends GtkWindow {
103
104    private $calculator;
105    private $display;
106    private $vbox;
107
108    function __construct($calculator) {
109       parent::__construct();
110       $this->calculator = $calculator;
111       $this->set_size_request(200, 240);
112       $this->connect_simple('destroy', array('Gtk','main_quit'));
113       $this->connect('key-press-event', array(&$this, "on_keypress"));
114       $this->connect('event', array($this, 'on_resize')); // note 1
115       $this->layout_widgets();
116    }
117
118    private function layout_widgets() {
```



```
119       // setup a vbox to hold display and buttons
120       $vbox = new GtkVBox();
121       $this->add($vbox);
122
123       // setup display
124       $this->display = new GtkEntry();
125       $this->display->set_alignment(1.0); // right-justified
126       $this->display->set_editable(false); // for display only
127       $vbox->pack_start($this->display);
128
129       // the button labels
130       $button_label = array(
131       array('Backspc', 'CE', 'C'),
132       array('7', '8', '9', '/', 'sqrt'),
133       array('4', '5', '6', '*', '%'),
134       array('1', '2', '3', '-', '1/x'),
135       array('0', '+/-', '.', '+', '=')
136       );
137
138       // setup the buttons
139       for ($j=0; $j<5; ++$j) {
140          $hbox = new GtkHBox();
141          $vbox->pack_start($hbox); // use a hbox to hold each row of buttons
142          for ($i=0; $i<5; ++$i) {
143             if ($j==0 && $i>2) continue; // first row contains only 3 buttons
144             $button = new CalculatorButton($button_label[$j][$i], $this->calculator);
145             $hbox->pack_start($button);
146          }
147       }
148    }
149
150    public function on_keypress($widget, $event) {
151
152       $keyval = $event->keyval;
153       $value = chr($keyval); // get the ASCII equivalent
154       if (preg_match("/[0-9\+\-\*\/\.=]/", $value) || $keyval==Gdk::KEY_Return ||
$keyval==Gdk::KEY_BackSpace) {
155          if ($keyval==Gdk::KEY_Return) $value='=';
156          if ($keyval==Gdk::KEY_BackSpace) $value='BackSpc';
157          $this->calculator->process_input($value);
158          $this->display->set_position(-1); // move the cursor to the end of display
159       }
160    }
161
162    public function set_display($value) {
163       print "value = $value\n";
164       $this->display->set_text($value);
165    }
166
167    public function on_resize($widget, $event) {// note 2
```



```
168         if ($event->type==2) { // note 3
169             $size = $this->get_size(); // note 4
170             $height = $size[1];
171             $new_font_size = intval(9 / 240 * $height); // note 5
172             $this->display->modify_font(new PangoFontDescription("$new_font_size")); //
    note 6
173         }
174     }
175 }
176
177 class CalculatorButton extends GtkButton {
178
179     private $calculator;
180
181     function __construct($label, $calculator) {
182         parent::__construct($label);
183         $this->calculator = $calculator;
184         $this->set_size_request(40, 32);
185         $this->connect('clicked', array($this, 'on_button'));
186
187         if (preg_match("/^([0-9]|\.)$/", $label))
188             $this->modify_bg(Gtk::STATE_NORMAL, GdkColor::parse('#FFFF33'));
189         if (preg_match("/^(\+|-|\*|\/){1}$/", $label))
190             $this->modify_bg(Gtk::STATE_NORMAL, GdkColor::parse('#ff99ff'));
191         if ($label=='C' || $label=='CE')
192             $this->modify_bg(Gtk::STATE_NORMAL, GdkColor::parse('#FF3366'));
193         if ($label=='=')
194             $this->modify_bg(Gtk::STATE_NORMAL, GdkColor::parse('#0099FF'));
195     }
196
197     function on_button($button) {
198         $value = $button->child->get_text();
199         echo "button_click: $value\n";
200         $this->calculator->process_input($value);
201     }
202 }
203
204 $cal = new Calculator();
205 $cal->main();
206
207 ?>
```

Listing 5.5.php

Explanation

1. Register signal ‘event’ on GtkWindow to be notified when user changes the window size.
2. Signal handler for signal ‘event’.

3. There are many events that will generate the signal ‘event’. In this case, we only want to process the signal ‘event’ generated by the left-mouse button.
4. Get the new window size.
5. Calculate the new display size.
6. Set the new display size by setting the font size.

5.6 Summary

We have reached the end of this book.

I hope the book has helped you unravel many of the “mysteries” you had with php-gtk2 before reading this book.

You should now have no problem understanding many of the examples in the php-gtk2 Cookbook website.

Writing php-gtk2 applications is as easy as:

- Laying out the widgets
- Connect the signal of interests
- Write the signal handlers

Along the way, you may want to make use of object-oriented programming to encapsulate related properties, methods and signal handlers using classes.

You have now a strong foundation to begin developing serious php-gtk2 applications.

I wish you good luck and all the best!

/kksou
January 2007

