Generating invoices with Glade

Pablo Dall'Oglio

Designing a GTK application interface by hand is not a very hard task and you may give your application a better peformance. However, it takes a lot of time, that as a professional developer you may be lacking of. Therefore, a graphical tool becomes necessary...

W SIECI

1. http://glade.gnome.org – the home page of the Glade project
2. http://http://gladewin32.sourceforge.net/ – Glade for Windows
3. *http://*

NA CD

For these purposes, there's Glade, a graphical tool built exactly to help the developers to design the interface of applications written in languages that use GTK (C, Python, PHP or Perl).

The sole purpose of the tool is making the interface building easier; it's not a code editor or IDE.

Using Glade, we can easily create windows, lists, buttons, boxes and organize them in a clear and simple way. At the end, the interface schema will be saved in a file *.glade*, that is an XML file containing the visual structure of the interface.

The basics of Glade

The Main window of Glade has the basic options (*Open* and *Save*), the Project Options and a list of windows containing our projects. The *Edit* menu has *Cut*, *Copy* and *Paste* options, while the *View* menu items allow to show or hide the pop-up windows (*Palette*, *Properties*, etc). See Figure 1.

The Palette

Each Glade project starts with a Window. The component that represents the window is `GtkWindow` and it's the first item of Glade's *Palette* (see Figure 2). After the first click on the *Windows* component, the first window comes up and then we can start using other widgets like `GtkHBox` (horizontal box), `GtkVBox` (vertical box), `GtkF` (allows absolute positions), `GtkL` (text label), `GtkEntry` (input bo-

O autorze

Pablo Dall'Oglio (pablo@dalloglio.net) is the author of the first book about PHP-GTK in the world. He is also the author of Agata Report (http://www.agatareports.com), Tulip Editor (http://tulip.solis.coop.br), as well as the Coordinator of GNUTeca project (http://www.gnuteca.com), an open source software for library management.

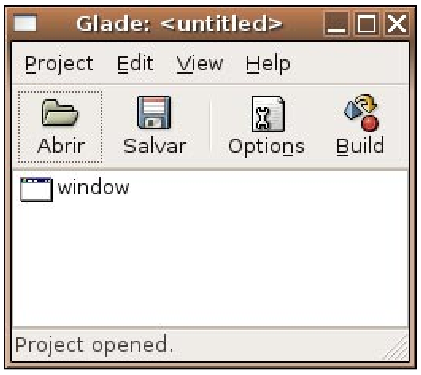


Figure 1. *The Main window of Glade*

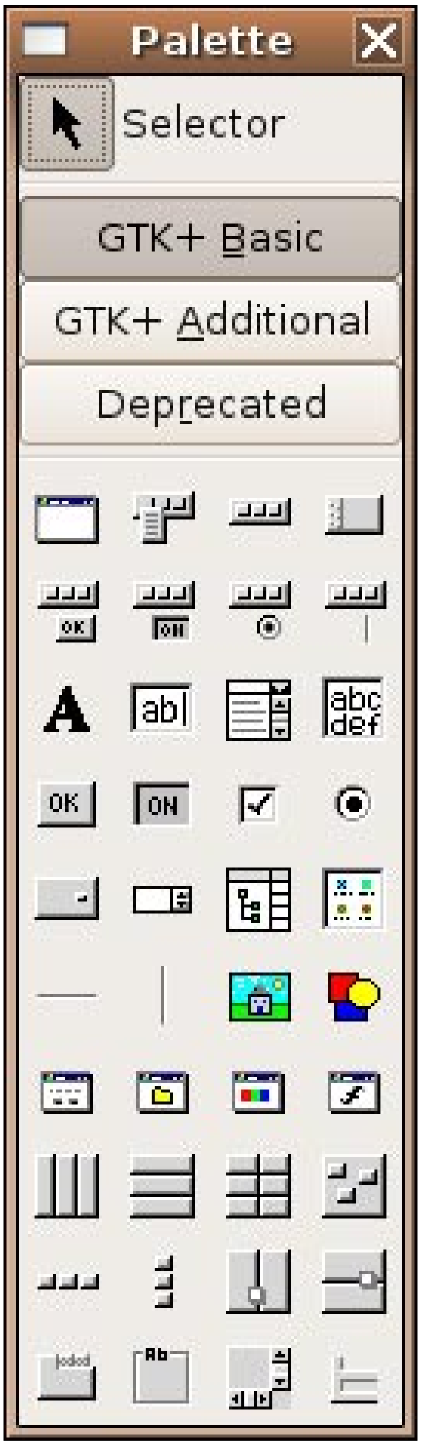


Figure 2. *Glade's Palette*

Installation

As a part of the GNOME environment, Glade should be present in the Linux distributions which contain GNOME. So, if you don't have it installed, search for the proper package in your distro first. If you won't find it there (or you would like to upgrade to the new version), get to the homepage of Glade (glade.gnome.org), download the sources and compile it in a standard way:

```
./configure
make
make install
```

Also, the Glade package should contain the INSTALL file, which provides more details on some non-typical installations.

In the Windows environment the things are very simple: just download Glade from *http://gladewin32.sourceforge.net* and install it.

Requirements

Glade needs GTK-2 installed. In order to use it with PHP, you will also need the PHP-GTK extension, which is available at the address of *http://gtk.php.net*. Furthermore, your PHP must be of the PHP5 branch.

xes), `GtkRadioButton`, `GtkCheckButton`, `GtkFrame`, `GtkImage`, `GtkComboBox`, `GtkToolBar`, etc.

The components list is separeted into pages (*Basic*, *Additional* and *Deprecated*). The *Deprecated* widgets nclude the old Gtk1 widgets (`GtkFileSelection`, `GtkCTree`, `GtkCList`, and so on).

The Widget Tree

The *Widget Tree* (*View* menu => *Show Widget Tree*) is a very nice tool that shows a hierarchy tree of all widgets you're using in your application. Each widget is a node in this tree (see Figure 3). The parent/child relationship tells us, which widget is inside another one. This feature is especially usefull to locate widgets, because many times the widget we want to change the properties of is not visible (as in the case of structural widgets like `GtkHBox`'es and `GtkVBox`'es). A simple right click on any widget in the tree allows you to select it. Then, you can go the the properties window, which content changes according to the selected widget.

The Project Window

We put our widgets into the *Project Window*. In the first window on the Figure 4 we have a `GtkWindow` with a `GtkFixed` component inside it. The `GtkFixed` component allows to put the components like buttons, labels, radiobuttons and checkbuttons in absolute coordinates inside the window. In the second window on the same figure we used a different aproach, putting a `GtkVBox` (Vertical Box) inside the window first, and then placing a `GtkHBox` (Horizontal Box) in the second row of the Vertical Box. The said Horizontal Box will then contain a GtkLabel in its second column and a GtkEntry in the fourth.

The Properties Window

The Properties window (see Figure 5) is the place where you set the properties of the selected widget. Its content depends on the widget you choose. Clicking on any widget on the screen causes this window to display properties of the class of the selected widget.

The Properties window has several tabs. The first one, called *Widget* defines

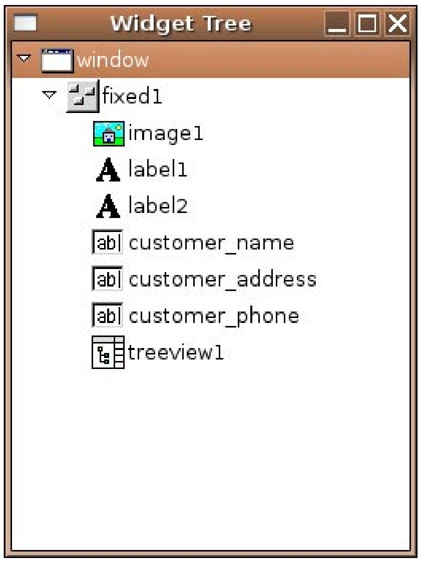


Figure 3. *Widget Tree*

Listing 1. *The Glade XML File*

```
<?xml version="1.0" standalone="no"?> <!--*- mode: xml -*-->
<!DOCTYPE glade-interface SYSTEM "http://glade.gnome.org/glade-2.0.dtd">
<glade-interface>
<widget class="GtkWindow" id="window1">
  <property name="visible">True</property>
  <property name="title" translatable="yes">window1</property>
  <property name="type">GTK_WINDOW_TOPLEVEL</property>
  <property name="window_position">GTK_WIN_POS_NONE</property>
  <property name="modal">False</property>
  <property name="resizable">True</property>
  <property name="destroy_with_parent">False</property>
  <property name="decorated">True</property>
  <property name="skip_taskbar_hint">False</property>
  <property name="skip_pager_hint">False</property>
  <property name="type_hint">GDK_WINDOW_TYPE_HINT_NORMAL</property>
  <property name="gravity">GDK_GRAVITY_NORTH_WEST</property>
  <property name="focus_on_map">True</property>
  <child>
    <widget class="GtkVBox" id="vbox1">
      <property name="visible">True</property>
      <property name="homogeneous">False</property>
      <property name="spacing">0</property>
      <child>
        <placeholder/>
      </child>
      <child>
        <widget class="GtkHBox" id="hbox2">
          <property name="visible">True</property>
          <property name="homogeneous">False</property>
          <property name="spacing">0</property>
          <child>
            <widget class="GtkLabel" id="label1">
              <property name="visible">True</property>
              <property name="label" translatable="yes">   Code:   </property>
              <property name="use_underline">False</property>
              <property name="use_markup">False</property>
              <property name="justify">GTK_JUSTIFY_LEFT</property>
              <property name="wrap">False</property>
              <property name="selectable">False</property>
              <property name="xalign">0.5</property>
              <property name="yalign">0.5</property>
              <property name="xpad">0</property>
              <property name="ypad">0</property>
              <property name="ellipsize">PANGO_ELLIPSIZE_NONE</property>
              <property name="width_chars">-1</property>
              <property name="single_line_mode">False</property>
              <property name="angle">0</property>
            </widget>
            <packing>
              <property name="padding">0</property>
              <property name="expand">False</property>
              <property name="fill">False</property>
            </packing>
          </child>
        </widget>
        <packing>
          <property name="padding">0</property>
          <property name="expand">True</property>
          <property name="fill">True</property>
        </packing>
      </child>
      <child>
        <placeholder/>
      </child>
    </widget>
  </child>
```

the basic properties of the selected widget. The most important feature of each widget is its name, because it's the property we refer the widget by in the application.

Other properties we can define in the *Properties* window are, for example, a label and align for a `GtkLabel`, a stock icon and a label for a `GtkButton`, the maximal length and the visibility of `GtkEntry`, and so on.

The second Tab (*Packing*) shown on the Figure 6 depends on the way you pack your widgets (in the image below, we have two modes). When the widget is packed inside a `GtkFixed` container, that allows absolute coordinates, it'll permit to change the widgets position (X and Y). When a widget is packed inside a `GtkVBox` or `GtkHBox` container, we'll be able to change other properties like *Expand* (if the widget will expand within the container's limit) or *Fill* (if the widget fills all of the containers' area), and so on.

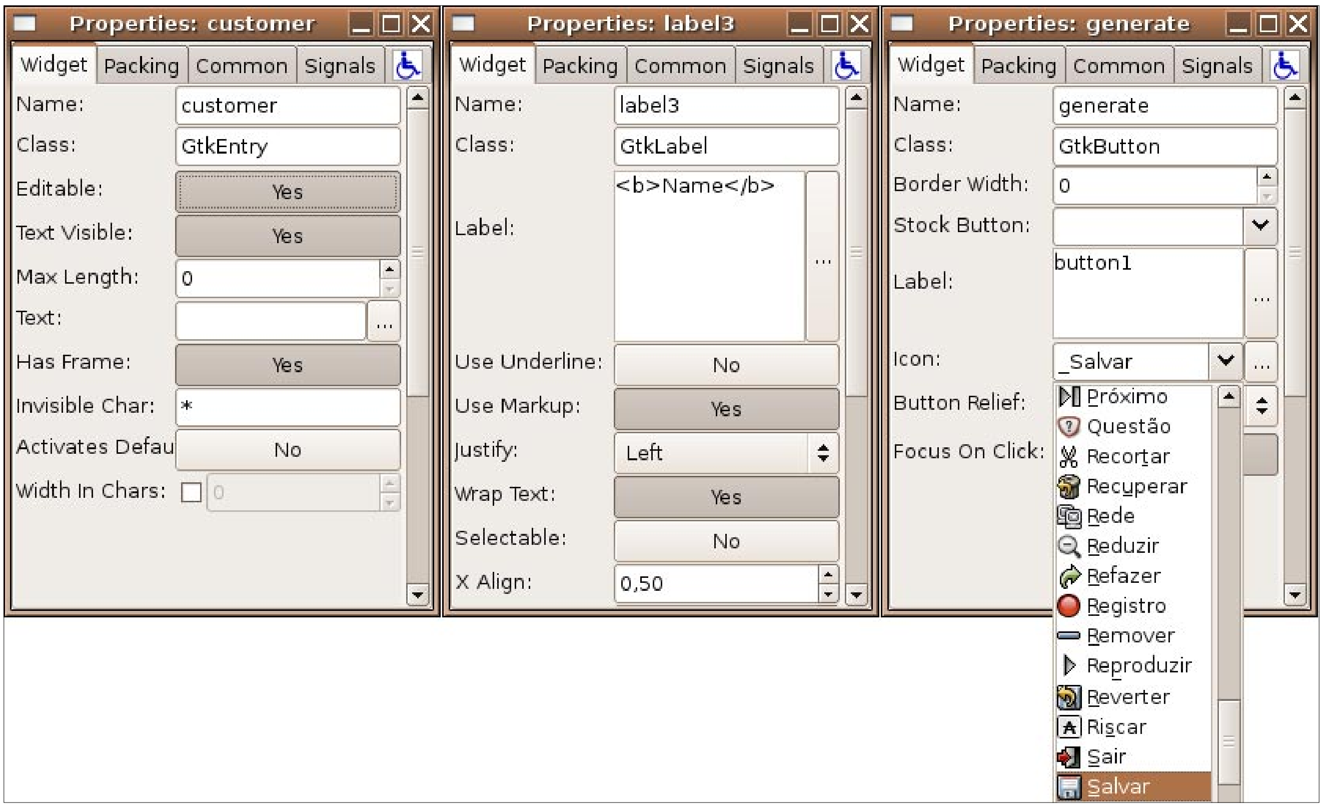
On the same Figure you will see the third Tab (*Common*), that contains shared properties among the widgets. Those include the features like *Width*, *Height*, visibility, ability of the widget to have a focus on, etc.

The Fourth Tab (*Signals*) allows you to connect the widget signals (`clicked`, `pressed`, `released` in the case of GtkButton) to a specific callback. This callback is a method or a function that must be included in the code of your application (it may be done later). When the widget emits a signal, the callback with the given name is executed. There's no need to connect the signals in glade, as it is being usually done later in the code.

The Glade XML File

When you save your file in Glade, it generates an XML file containing the the visual structure of the project (Listing 1). It contains the names of the widgets, their properties like dimensions, labels, icons, the way each widget will be packed, all registered callbacks and more. The XML tags are used to define the properties of each widget, and the hierarchy of tags represents containers and their children.

Then, the Glade project file can be used in your application. It has to be processed through the `GladeXML` class that parses the XML content, making the widgets available to our application. For the latter purpose we use a method named `get_widget()`. The only thing we have to



```
Properties: customer
Widget | Packing | Common | Signals
Name:              customer
Class:             GtkEntry
Editable:          Yes
Text Visible:      Yes
Max Length:        0
Text:
Has Frame:         Yes
Invisible Char:    *
Activates Defau    No
Width In Chars:    0

Properties: label3
Widget | Packing | Common | Signals
Name:              label3
Class:             GtkLabel
Label:             <b>Name</b>
Use Underline:     No
Use Markup:        Yes
Justify:           Left
Wrap Text:         Yes
Selectable:        No
X Align:           0,50

Properties: generate
Widget | Packing | Common | Signals
Name:              generate
Class:             GtkButton
Border Width:      0
Stock Button:
Label:             button1
Icon:              _Salvar
Button Relief:
Focus On Click:
    Próximo
    Questão
    Recortar
    Recuperar
    Rede
    Reduzir
    Refazer
    Registro
    Remover
    Reproduzir
    Reverter
    Riscar
    Sair
    Salvar
```

Figure 5. *The Properties window for GtkEntry, GtkLabel and GtkButton*

do is to know the name of each widget we want to retrieve from the Glade project file.

The first example

In our first practical example, we'll design a window (`GtkWindow`) with a vertical box inside (`GtkVBox`). In the first position of the vertical box, we'll place a text label, which will contain the *Type your Name* phrase in bold. The second position will be an input entry called `name`. In the third one, we'll put a button named *Print*, while the fourth position will contain another text label of the name `result`. See the Figures 7 and 8.

The program works this way: the user can type anything inside the `name` input text. When he clicks on the `Print` button, the text `Hello <name>` is set in the result text label, where `<name>` is the expression typed in the `name` field. See the Figure 9.

This example is very simple, but it's very good to illustrate the essential abilities of Glade and joining the generated interface with our application. Pay attention on how we reuse the file designed in Glade. On the Listing 2 we show the code that allows this. First we instantiate an object of the `GladeXML` class as `$glade`. Then, using the object's method named `get_widget()` we get three widgets we created in Glade: `name`, `print` and `result` and create three objects: `$name`, `$print` and `$result`, respectively. Then, using those objects we can do anything with the widgets they represent: change their properties, connect their signals, show or hide them, and so on.

Listing 2. *Using the Glade file in a PHP application*

```
<?php
# instantiate the glade object
$glade  = new GladeXML('exemplo1.glade');
# get the name input entry
$name   = $glade->get_widget('name');
# get the print button
$print  = $glade->get_widget('print');
# get the result label
$result = $glade->get_widget('result');
# connect the print button
$print->connect_simple('clicked', 'onPrint');
function onPrint(){
    global $name, $result;

    # get the typed text
    $text = $name->get_text();
    # set the result label
    $result->set_text("Hello {$text}");
}
Gtk::Main();
?>
```

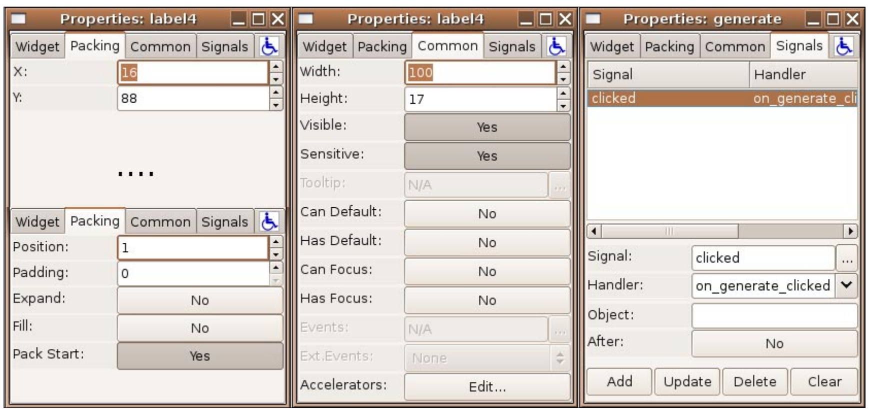
The Application

It's time to create our main application: The Invoice Generator. We'll basically use Glade to design the interface. Then we'll create the source code that will make use of our generated interface (as shown in the previous example). It will also employ the FPDF library for the purpose of generating the invoices in the PDF format.

FPDF is an open source library which allows to generate PDF files with pure PHP. It has lots of facilities to draw images, lines, text and so on.

The Main Window

We'll start with designing the main window



Properties: label4

Widget | Packing | Common | Signals

X: 16

Y: 88

....

Widget | Packing | Common | Signals

Position: 1

Padding: 0

Expand: No

Fill: No

Pack Start: Yes

Properties: label4

Widget | Packing | Common | Signals

Width: 100

Height: 17

Visible: Yes

Sensitive: Yes

Tooltip: N/A

Can Default: No

Has Default: No

Can Focus: No

Has Focus: No

Events: N/A

Ext.Events: None

Accelerators: Edit...

Properties: generate

Widget | Packing | Common | Signals

Signal	Handler
clicked	on_generate_cli

Signal: clicked

Handler: on_generate_clicked

Object:

After: No

Add | Update | Delete | Clear

Figure 6. *The Properties window: Packing, Common and Signals tabs*

of the interface of our application (see Figure 10), which is the first element of the palette shown on the Figure 11. As soon as you click on the GtkWindow button, the first window will appear. Then we can start placing widgets inside the window. `GtkWindow` accepts just one widget inside it, so you must care about a kind of widget you add: it must be a container, and this means widgets that inherit from the `GtkContainer` base class. Therefore, you can use, among others, vertical boxes (`GtkVBox`), Horizontal boxes (`GtkHBox`), frames (`GtkFrame`) or the widget named `GtkFixed`. We'll use the last one, as it allows us to position widgets with absolute coordinates into its area.

Adding an Image

Let's begin placing widgets in our main window. The first one will be a logo. We'll do this using the button of the palette that looks like a drawing and represents the `GtkImage` widget. We can load almost all popular formats, including PNG, JPG, XPM and many others. `GtkImage` also supports transparency. So, let's just choose GtkImage from the Palette, and click again at the position inside the `GtkFixed` area you want to place it at. Then we change the `icon` property of the `GtkImage` to select an image file (see Figure 13). There's not much to do with the image besides adjusting its position on the screen, that we can achieve using the mouse or editing the properties of the *Packing* tab.

Adding a Label

The next thing to do is to add the text labels (`GtkLabel`). One of the great news in GTK-2 is the Pango framework which makes enables us to use a markup language to define fonts, sizes, color and styles, because it adds lots of flexibility to the presentation of text.

To put a `GtkLabel` component inside our `GtkFixed` area, we just need to click at the `GtkLabel` component and then click at the place we want to put it. Then, we can start editing its `Label` property, where we can use the new markup language (remember to turn on the button *Use Markup*). See the Figure 15.

Using the definition `<span font_desc="Times Bold Italic 10">` we define the font attributes. Using `<span foreground=red>` we set the color, and

Listing 3. *Invoice generator in PHP (Invoice.php)*

```
<?
/**
* Class Invoice
* Deals with interface and starts the invoice generation
*/
final class Invoice extends GladeXML{
   private $invoice;            // invoice object
   private $taxes        = 0.05; // 5%
   private $shipping_fee = 0.4;  // $ 0.4 per unity
   private $amount       = 0;    // quantity of items
   /**
    * Constructor Method
    * Reads the interface and connects the signals
    */
   public function __construct(){
      // Call GladeXML Constructor Method
      parent::__construct('interface.glade');
      // define the standard for calculations
      setlocale(LC_ALL, 'POSIX');
      // Connect the action buttons
      $this->saveButton->connect_simple('clicked',array($this,'saveInvoice'));
...
      // define the invoice date as today
      $this->invoiceDate->set_text(date("Y-m-d"));
      // creates the model to store the data
      $this->model = new GtkListStore(Gtk::TYPE_STRING, Gtk::TYPE_STRING,
         Gtk::TYPE_STRING, Gtk::TYPE_STRING);
      $this->itemsList->set_model($this->model);
      // creates the columns
      $column1 = new GtkTreeViewColumn();
...
      // create the cell renderers
      $cell_renderer1 = new GtkCellRendererText();
...
      // pack one cell renderer per column
      $column1->pack_start($cell_renderer1, true);
...
      // define the indexes
      $column1->set_attributes($cell_renderer1, 'text', 0);
...
      $this->itemsList->append_column($column1);
...
      // define the titles of the columns
      $column1->set_title('Code');
...
    }
   /**
    * Method __get()
    * Returns a glade object whenever a property doesn't exist
    */
   public function __get($property){
      // retrieve the object from glade
      return parent::get_widget($property);
```

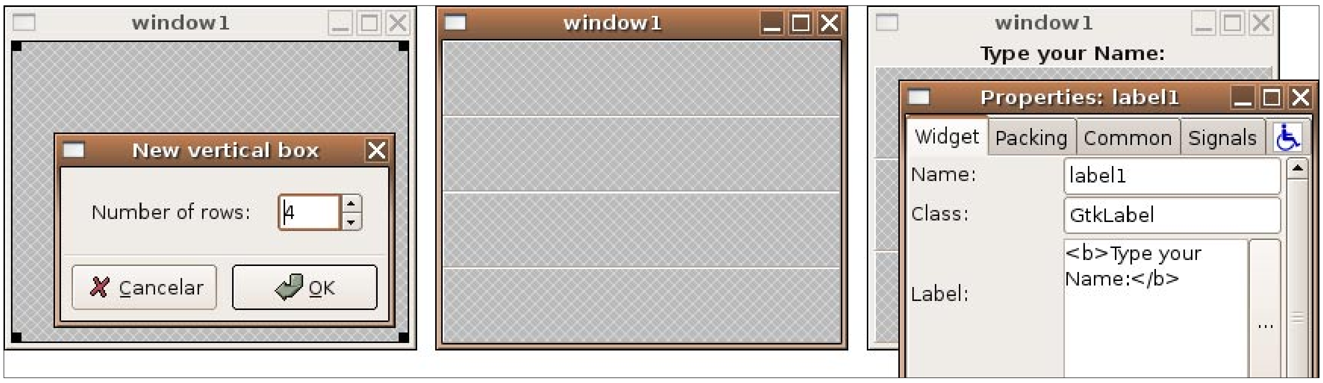


Figure 7. *Making our first example*

Listing 4. *Invoice generator in PHP (Invoice.php), continued*

```
    /**
     * Method addItem()
     * Add an Item to the list
     */
    public function addItem(){
       $product    = array(); // initiates the product array
       // get the data from screen
       $product[] = $this->productCode->get_text();
...
       // add the product to the model
       $this->model->append($product);
       // increments the amount
       $this->amount += $this->productQuantity->get_text();
       // calculates the subtotal
       $this->subTotal->set_text($this->subTotal->get_text() +
                                  $this->productPrice->get_text());
       // calculates the shipping cost
       $this->Shipping->set_text($this->amount * $this->shipping_fee);
       // calculates the taxes
       $this->Taxes->set_text($this->subTotal->get_text() *
                              $this->taxes);
       // calculates the total
       $this->Total->set_text($this->subTotal->get_text() +
          $this->Shipping->get_text() +
          $this->Taxes->get_text());
       // clear the product data
       $this->productCode->set_text('');
...
       // focuses the product code
       $this->window->set_focus($this->productCode);
    }
    /**
     * Method clearItems()
     * clear the Items list
     */
    public function clearItems(){
       // clear the items
       $this->model->clear();
       $this->amount = 0;
       // clear the subtotals,...
       $this->subTotal->set_text(0);
...
       // increments the invoice's number
       $this->invoiceNumber->set_text($this->invoiceNumber->get_text() + 1);
    }
    /**
     * Method saveInvoice()
     * Generate the invoice
     */
```

using `<b>`, `<i>` or `<u>` we define the style (bold, italic or underline). This markup language is very close to HTML. It is described in detail (both Pango API and all keywords we can use) at the address of *http://developer.gnome.org/doc/API/2.0/pango/PangoMarkupFormat.html*.

This way, we'll add the labels containing the texts *Customer*, *Name*, *Address*, *Invoice*, *Date*, *Product*, *Code*, *Description*, *Quantity*, *Price*, *SubTotal*, *Taxes*, *Shipping* and *Total*.

Adding a Text Entry

The next step is to add the text entry (`GtkEntry`) widgets. We'll start with the one called `customerName` (Figure 17). Then we'll adjust its absolute position using the *Packing* properties and its size, using the *Common* features. We'll repeat this step for other text entries: `customerAddress`, `customerPhone`, `productCode`, `productDescription`, `productQuantity`, `productPrice`, `subTotal`, `Taxes`, `Shipping`, `Total`, `invoiceNumber` and `invoiceDate`.

Adding a Button

Now, let's place some action buttons in our interface. The first one will use a stock image (`+Add`) and will be called `addButton`. This button will be responsible for reading the product information: code, description, quantity and price from the text entries and adding these data to a list of items, that will be created in the next step. The interface we'll have another action buttons, but there's no need to replicate the explanation for each button. We'll explain all the interface ahead giving names for all the widgets on the window.

Adding the TreeView

Finally, we'll add a GtkTreeView widget. `GtkTreeView` is responsible for showing both lists and trees. One of the most

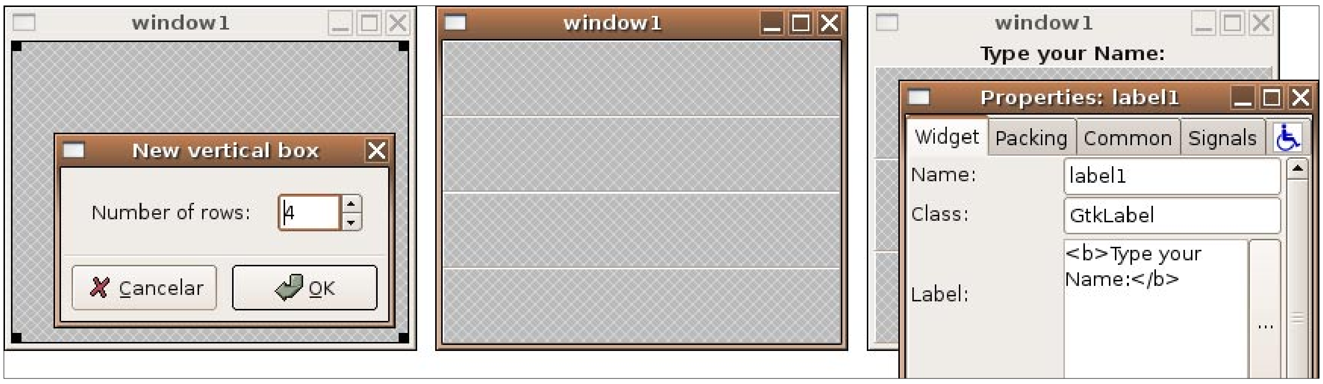


Figure 7. *Making our first example, continued*

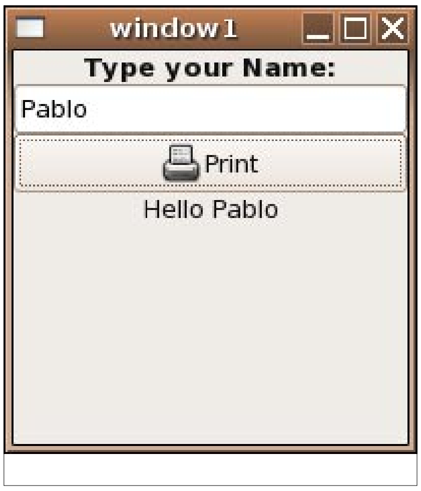
important features of `GtkTreeView` is the complete separation of the data model from its visualization. So Glade can place an empty `GtkTreeView` on the screen (Figure 21), but the data model can only be created inside our code, using the structures (strings, arrays) of the choosen language (in our case, PHP). We will use `GtkTreeView` in the list mode, and then add the columns inside our application.

The Main Interface

Our interface is complete, we present it on the Figure 22 (in the edition mode) and on the Figure 23 (as seen in the working application). In the header, we placed the logo of the Gnome Foundation, the title of the program and the address, formatted using the new Pango markup language. Also, we have the *Invoice Number* and the *Date*. In the *Customer* section we have the labels and text entries for the *Customer's Name*, *Address* and *Telephone*. In the

Listing 5. *Invoice generator in PHP (Invoice.php), continued*

```
   public function saveInvoice(){
      // include our class for Invoice Generation
      include_once('InvoicePdf.class.php');
      // instantiates the invoice object
      $this->invoice = new InvoicePdf();
      // define the invoice's number and date
      $this->invoice->setNumber($this->invoiceNumber->get_text());
      $this->invoice->setDate($this->invoiceDate->get_text());
      // creates the customer object
      $customer->name     = $this->customerName->get_text();
...
      // define the invoice's customer
      $this->invoice->setCustomer($customer);
      // loop all the products
      $iter = $this->model->get_iter_first();
      while ($iter){
         $code        = $this->model->get_value($iter, 0);
...
         // add the product data to our invoice's object
         $this->invoice->addItem($code, $description, $quantity, $price);
         $iter = $this->model->iter_next($iter);
      }
      // define the footer information
      $this->invoice->setFooter($this->subTotal->get_text(),
...
      $this->Total->get_text());
      // ask the user to save the file
      $dialog = new GtkFileChooserDialog('Saving...', NULL,
         Gtk::FILE_CHOOSER_ACTION_SAVE,array(Gtk::STOCK_OK,Gtk::RESPONSE_OK,
         Gtk::STOCK_CANCEL, Gtk::RESPONSE_CANCEL));
      $response = $dialog->run();
      if ($response == Gtk::RESPONSE_OK){
         // generates the invoice, passing the filename
         $this->invoice->Generate($dialog->get_filename());
      }
      $dialog->destroy();
      $this->clearItems();
   }
}
// instantiate the interface
$application = new Invoice;
Gtk::Main();
?>
```



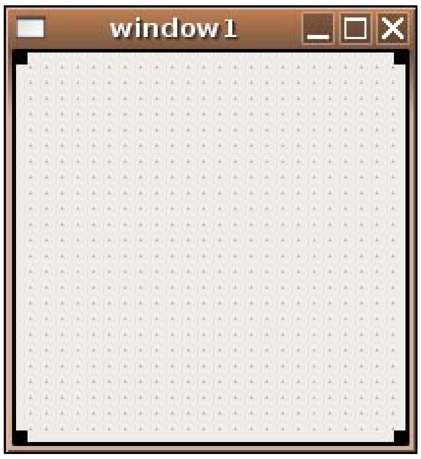


Figure 10. *An empty editor window*

Product section we have the analogical labels and entries for the *Product Code*, *Description*, *Quantity* and *Price*, besides the *Add* button wich reads the product data and adds it to the items list. In the bottom-right corner we have *SubTotal*, *Taxes*, the *shipping* costs and the *Total* field that are automatically calculated. The taxes are 5% and the shipping cost is $0,40 per product unit. Each widget we present on this Figure has its name written in red and in the triangle brackets <>. Of course, those names won't be visible in the working application.

Let's make the code: the Main File

On the Listings 3, 4 and 5 we present the main file containing our PHP code (*invoice.php*). We start defining the class `invoice`, that will extend the `GladeXML` class. Let's take a look at its constructor method. First, we call the constructor of the class' parent (GladeXML), passing the name of the file containing our interface, *interface.glade* to the constructor as an argument. Then we define the POSIX standard for calculations.

The next important thing to do is to connecgt the action buttons with the proper callback functions. For this purpose, we will use the `connect_simple()` method of each button: `$this->saveButton`, `$this->clearButton` and `$this->addButton`, which serve for saving data, clearing the

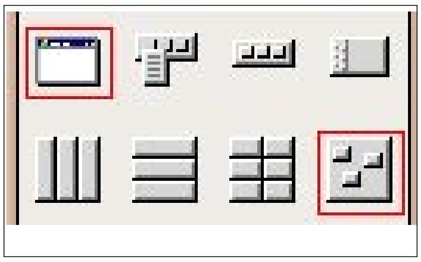
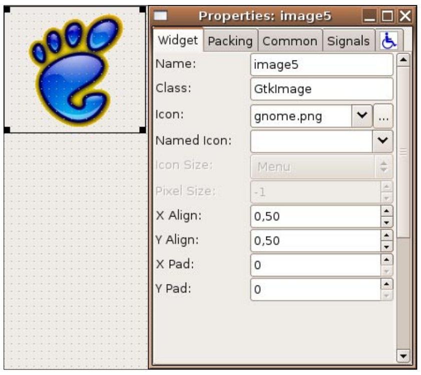


Figure 9. *The complete example*



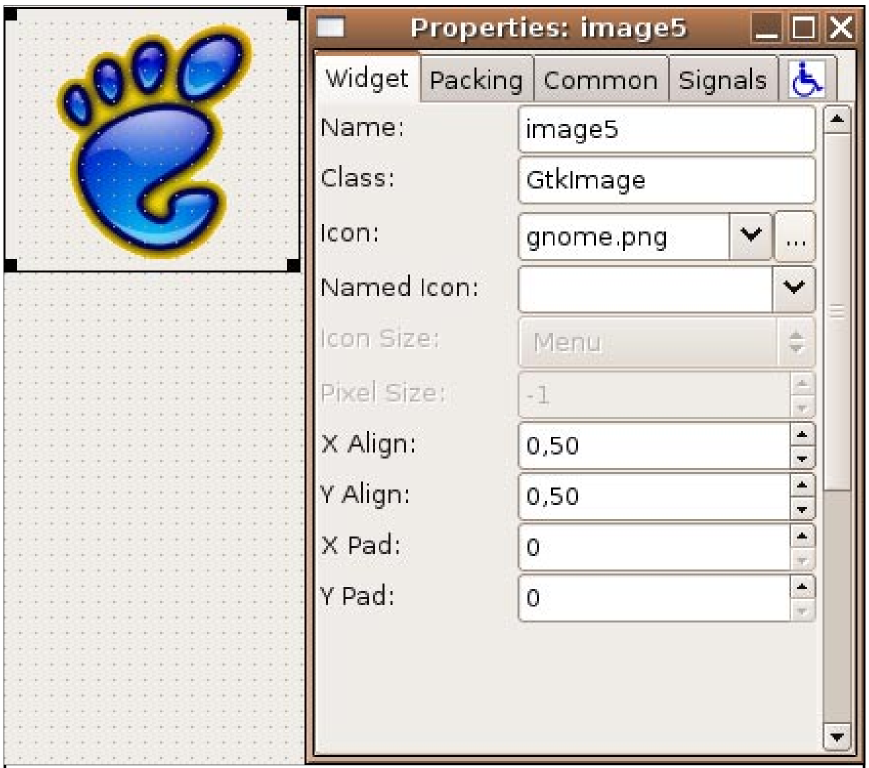
Name: image5
Class: GtkImage
Icon: gnome.png
Named Icon:
Icon Size: Menu
Pixel Size: -1
X Align: 0,50
Y Align: 0,50
X Pad: 0
Y Pad: 0

Figure 13. *Glade's palette*

Listing 6. *The class responsible for the PDF generation (InvoicePdf.class.php)*

```
<?
/**
* Class InvoicePdf
* Genereates the Invoice with the FPDF library
*/
class InvoicePdf{
    private $pdf;       // Instance of FPDF class
    private $number;    // Invoice Number
    private $customer;  // Customer Object
    private $items;     // array with the products
    /**
     * Constructor Method
     * Creates the PDF document
     */
    public function __construct(){
        // Define the fonts directory of FPDF
        define('FPDF_FONTPATH', getcwd() . '/fpdf/font/');
        // Load the FPDF library
        include_once 'fpdf/fpdf.php';
        // creates a new PDF document
        $this->pdf = new FPDF('P', 'pt', array(596,540));
        $this->pdf->SetMargins(2,2,2); // define margins
    }
    /**
     * Method setNumber
     * Define the Invoice Number
     */
    public function setNumber($number){
        $this->number = $number;  // define the invoice's number
    }
    /**
     * Method setDate
     * Define the Invoice Date
     */
    public function setDate($date){
        $this->date = $date;  // define the invoice date
    }
    /* Method setCustomer
     * Store an object with the customer's data
     */
    public function setCustomer($object){
        $this->customer->name    = $object->name;
...
    }
   /* Method setFooter
    * Define the totals, subtotals, taxes...
    */
   public function setFooter($subTotal, $Taxes, $Shipping, $Total){
      $this->subTotal = $subTotal;
...
   }
   /* Method addItem
   * Add the product's data to the array of items
   */
   public function addItem($code, $description, $quantity, $price){
      $this->items[] = array($code, $description, $quantity, $price);
   }

/* Method Generate
    * Generates the PDF Document and saves it
    * to the file passed as the first parameter
    */
   public function Generate($archive){
      // create an empty page
      $this->pdf->AddPage();
      $this->pdf->Ln();
      $image  = 'gnome.jpg';
```



Name: image5
Class: GtkImage
Icon: gnome.png
Named Icon:
Icon Size: Menu
Pixel Size: -1
X Align: 0,50
Y Align: 0,50
X Pad: 0
Y Pad: 0

Figure 15. *Glade's palette*

Listing 7. *The class responsible for the PDF generation (InvoicePdf.class.php), continued*

```
         $size   = getimagesize($image);
      $width  = $size[0];
      $height = $size[1];
      // draw the image at the x=7,y=7
      $this->pdf->Image($image, 7, 7, $width, $height);
      $this->pdf->SetLineWidth(1);
      $this->pdf->SetFillColor(247,247,247);
      $this->pdf->Rect(434,24,147,57, 'DF');
      // print Company's name and address. Print invoice number and date.
      $this->pdf->SetTextColor(0,0,0);
      $this->pdf->SetFont('Times','B',28);
      $this->pdf->Text(500,48, $this->number);
      $this->pdf->Text(140,48, 'Gnome Foundation');
...
      // print company's address, customer's data and the product's headers
...
      $row=244;
      if ($this->items){
         // print product by product
            foreach ($this->items as $item){
               $this->pdf->Text(14, $row, $item[0]);
...
               $this->pdf->Text(400,$row, $item[2]);
               $this->pdf->Text(500,$row, 'U$ ' . number_format($item[3], 2));
               $row += 16;
            }
      }
      $this->pdf->SetFillColor(247,247,247);
      $this->pdf->Rect(7,416,580,80, 'DF');
      $this->pdf->SetFont('Courier','B',14);
      // print the invoice's footer with totals and fees...
      $this->pdf->Text(440, 434, 'SubTotal: ' . $this->subTotal);
      $this->pdf->SetTextColor(222, 0, 0);
...
      $this->pdf->Output($archive);
      exec("gpdf $archive &"); // opens the PDF with your favorite viewer!
```

entries and adding a new item to the list, respectively. Later, we will create the data model for the list of items.

It's a pain to retrieve an object from the glade file every time we need it. So, to make the life easier we'll use a new resource from PHP5, the magic method `__get()`. This method intercepts all attempts to access an object property in order to read its value.

We will use this method to determine if someone demands an access to the property that doesn't exist. In such a case, PHP instantiates an object through the `get_widget()` method of the `GladeXML` class.

Therefore, when we need to access the `customerName` object from the glade interface, we just access `$this->customerName`. The same happens, when we have to use the *Save* button: we just use `$this->saveButton`.

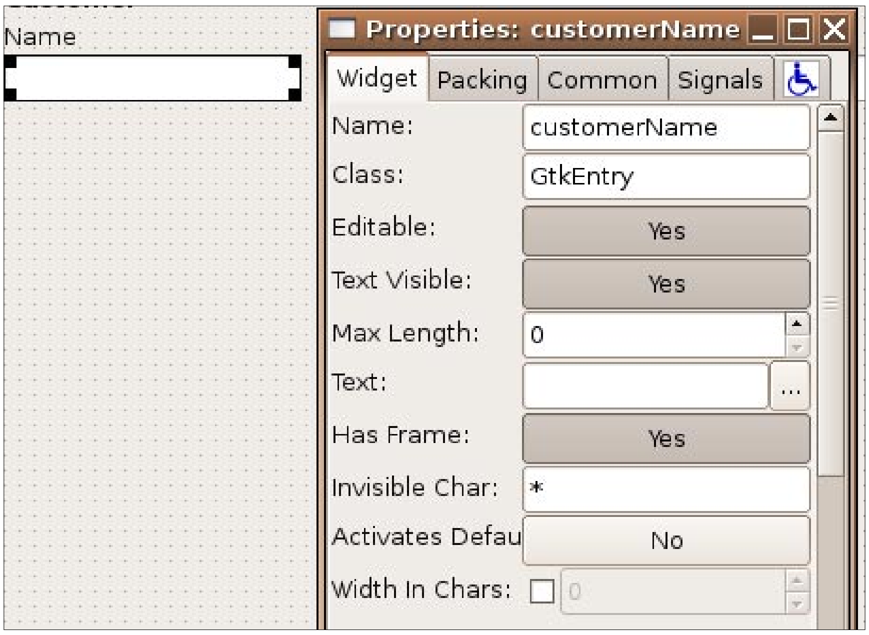
Connecting of signals lets the user of our application to make use of the buttons. Whenever the users inputs the product information and clicks on the button *Add* ,the method `addItem()` is executed, adding the product information to the item list. When the user clicks on the *Save* button, PHP calls the method `saveInvoice()` and the user is asked for the filename through the `GtkFileChooserDialog` class. Then, the class `InvoicePdf`, responsible for the PDF creation, is instantiated, resulting in the PDF file being generated. After generating invoice, the invoice number is incremented and the list of items is cleared for the next invoice.

The PDF Generation

This class is responsible for the PDF creation (Listings 6 and 7). It instantiates the FPDF class and makes available some methods like `setCustomer()` that defines the customer's data (name, address and phone), `setFooter()` that defines the footer's information (subtotal, shipping costs, taxes and total), `addItem()` that adds a product to the list of products and `Generate()` that reads all the information and draw the PDF document using the FPDF library. After that, it opens the result in the favorite viewer.

The Invoice

Here we are with the result: our Invoice (see the Figure 24). It's a PDF file containing the company's logo, name and address at the left top corner and the



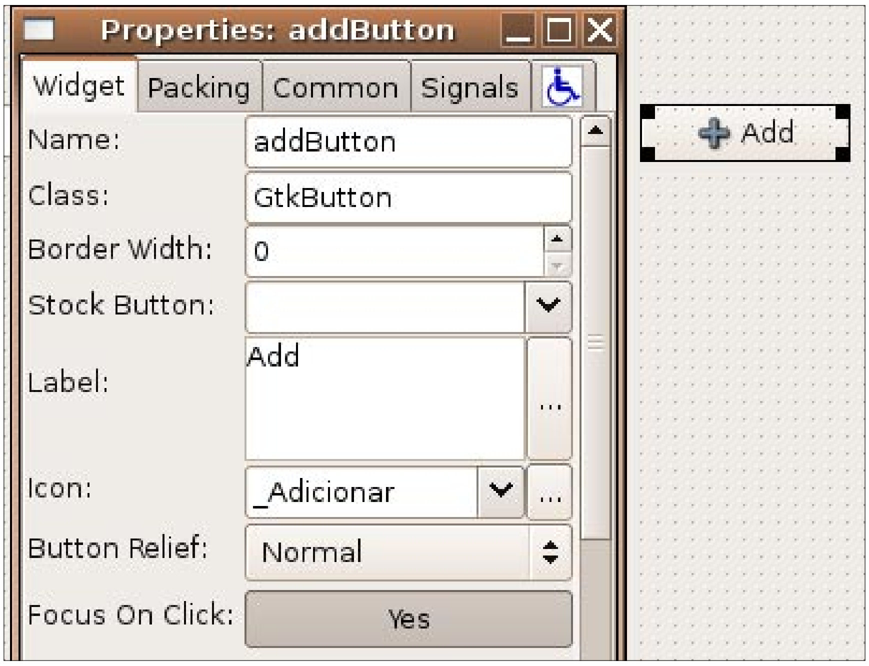
Name

Properties: customerName

Widget | Packing | Common | Signals

Name:	customerName
Class:	GtkEntry
Editable:	Yes
Text Visible:	Yes
Max Length:	0
Text:	
Has Frame:	Yes
Invisible Char:	*
Activates Defau	No
Width In Chars:	0

Figure 17. *Properties of the customerName text entry (GtkEntry)*

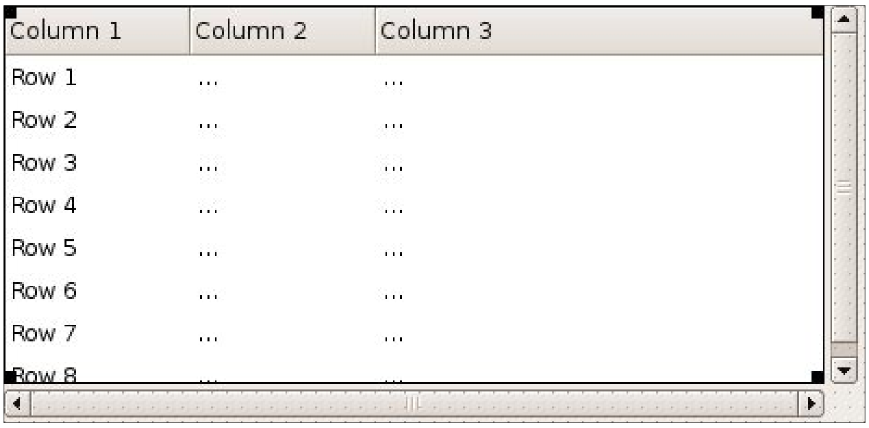


Properties: addButton

Widget | Packing | Common | Signals

Name:	addButton
Class:	GtkButton
Border Width:	0
Stock Button:	
Label:	Add
Icon:	_Adicionar
Button Relief:	Normal
Focus On Click:	Yes

Figure 19. *Properties of the addButton button (GtkButton)*



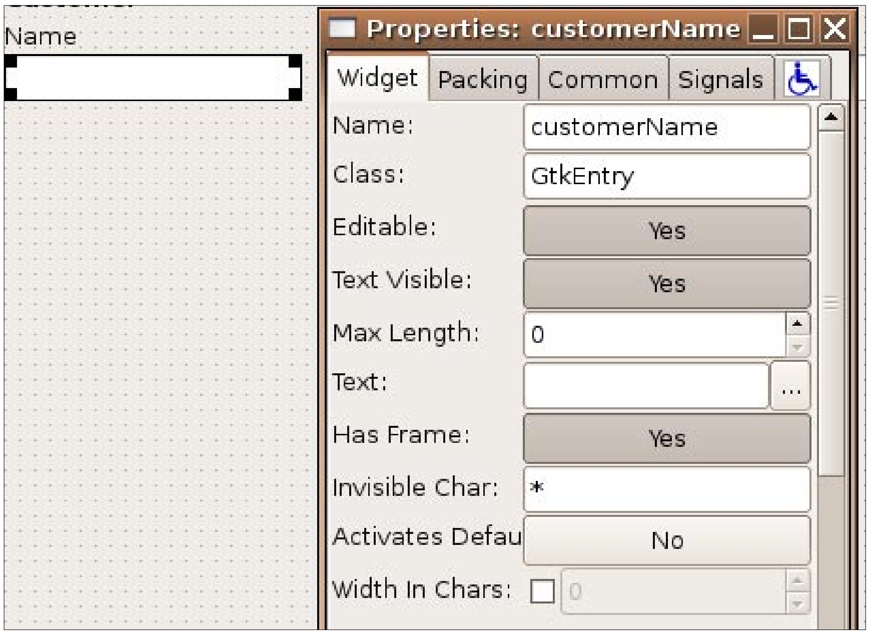
Column 1	Column 2	Column 3
Row 1	...	...
Row 2	...	...
Row 3	...	...
Row 4	...	...
Row 5	...	...
Row 6	...	...
Row 7	...	...
Row 8	...	...

Figure 21. *GtkTreeView widget in the List mode*

invoice number and date at the right top corner with a rectangle around. Below, we have the *Customer* info (*Name*, *Address* and *Phone*) inside a gray rectangle, and in the middle of our Invoice we have the *Products* list (*Code*, *Description*, *Quantity* and *Price*) with a header containing the names of the columns inside a dark gray rectangle. At the footer we have the *SubTotal*, *Taxes*, *Shipping* and the *Grand Total* with the colors identical to those used used in the Glade interface.

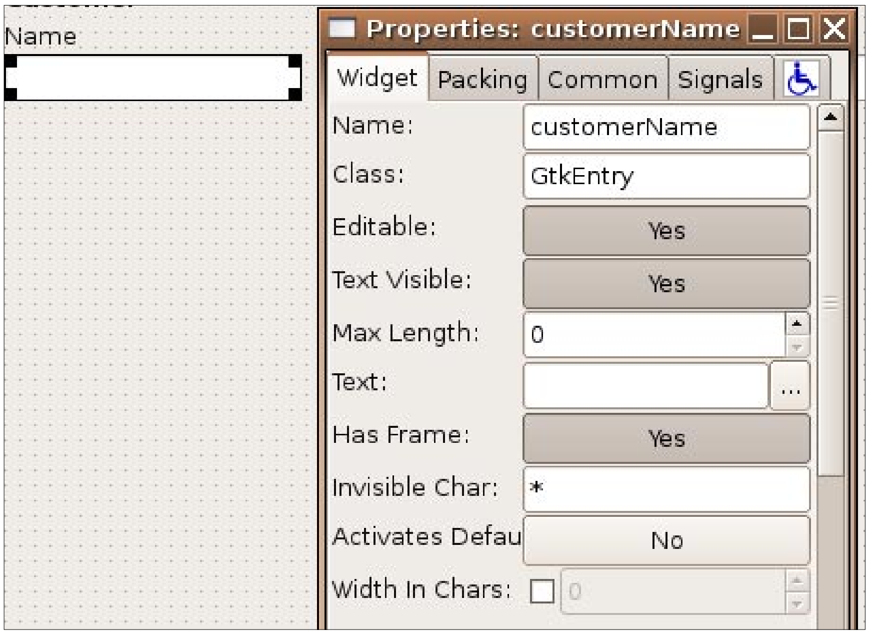
Summary

Our Invoice Generator is ready and working, proving you, how easy it is to make the Graphical User Interfaces using Glade. Also, it dispells the myths telling, that PHP is suitable for the server-side purposes only. Joining PHP5, PHP-GTK and Glade is a very good choice for any professional software developer who wants to create the Graphical User interfaces using the widely accepted, tested technologies and maintain flexibility of his (or her) projects.



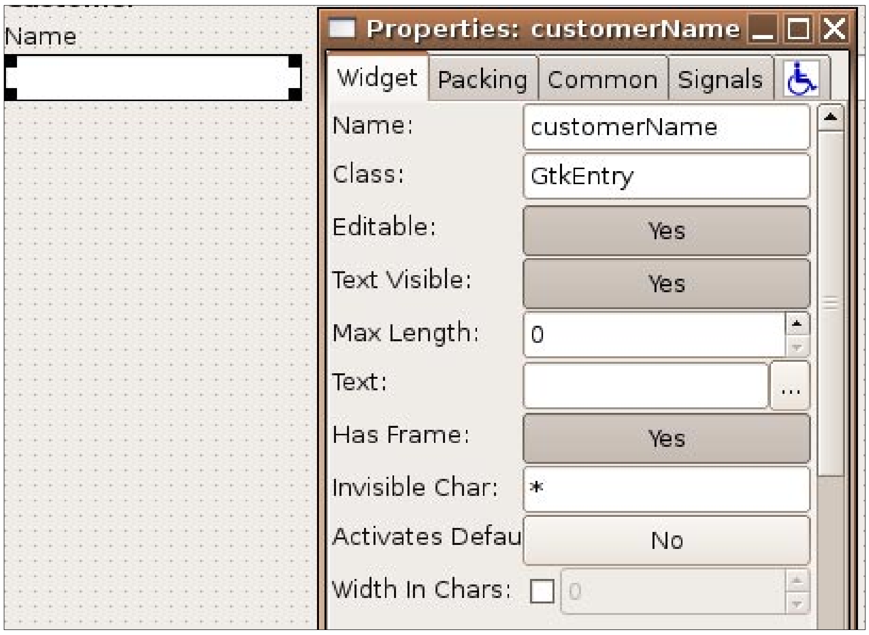
Name: customerName
Class: GtkEntry
Editable: Yes
Text Visible: Yes
Max Length: 0
Text:
Has Frame: Yes
Invisible Char: *
Activates Defau No
Width In Chars: 0

Figure 22. *The complete interface as seen in the editor*



Name: customerName
Class: GtkEntry
Editable: Yes
Text Visible: Yes
Max Length: 0
Text:
Has Frame: Yes
Invisible Char: *
Activates Defau No
Width In Chars: 0

Figure 23. *The complete interface as seen in the working application*



Name: customerName
Class: GtkEntry
Editable: Yes
Text Visible: Yes
Max Length: 0
Text:
Has Frame: Yes
Invisible Char: *
Activates Defau No
Width In Chars: 0

Figure 24. *The generated invoice*