

# **wxGlade user manual**

# Contents

<b>1</b>	<b>Introduction to wxGlade</b>	<b>1</b>
1.1	What is wxGlade? . . . . .	1
1.2	What Can You Do with wxGlade? . . . . .	2
1.3	What is wxGlade NOT? . . . . .	2
1.4	Basics . . . . .	2
1.5	Requirements and Supported Platforms . . . . .	2
1.6	Download . . . . .	2
1.7	Installation . . . . .	3
1.7.1	Installing at Microsoft Windows . . . . .	3
1.7.2	Installing at Unix/Linux . . . . .	3
1.7.3	Installing from Source . . . . .	3
1.7.3.1	Single user installation . . . . .	3
1.7.3.2	Multi user installation - variant 1 . . . . .	3
1.7.3.3	Multi user installation - variant 2 . . . . .	3
<b>2</b>	<b>Exploring wxGlade</b>	<b>5</b>
2.1	Quick start . . . . .	5
2.2	Basics of wxGlade . . . . .	6
2.3	Best Practice . . . . .	6
2.4	Command line invocation . . . . .	6
2.5	Using the source code . . . . .	7
2.5.1	Full control by wxGlade . . . . .	7
2.5.2	Shared control . . . . .	8
2.5.3	Output path and filenames . . . . .	8
2.5.4	Automatically created wxApp instance . . . . .	8
2.5.5	Compiling C++ code . . . . .	9
2.6	Handling XRC files . . . . .	9
2.7	Specifying the path of bitmaps . . . . .	10

<b>3</b>	<b>wxGlade User Interface</b>	<b>11</b>
3.1	Main Palette . . . . .	11
3.2	Tree Window . . . . .	11
3.3	Design Window . . . . .	13
3.4	Properties Window . . . . .	13
3.4.1	Common Properties . . . . .	14
3.4.2	Layout Properties . . . . .	20
3.4.3	Widget Properties . . . . .	20
3.4.4	Events Properties . . . . .	21
3.4.5	Code Properties . . . . .	23
3.5	Preferences Dialog . . . . .	24
3.6	The wxGlade Menu . . . . .	24
3.6.1	The FILE menu . . . . .	25
3.6.2	The VIEW menu . . . . .	25
3.6.3	The HELP menu . . . . .	25
3.7	Shortcuts . . . . .	25
<b>4</b>	<b>Supported widgets</b>	<b>27</b>
4.1	Introduction . . . . .	27
4.2	Widget list . . . . .	27
4.2.1	Frame . . . . .	27
4.2.2	Dialog or Panel . . . . .	27
4.2.3	Panel . . . . .	27
4.2.4	Splitter window . . . . .	27
4.2.5	Notebook . . . . .	28
4.2.6	Button . . . . .	28
4.2.7	Toggle button . . . . .	28
4.2.8	Bitmap button . . . . .	28
4.2.9	Text control . . . . .	28
4.2.10	Spin control . . . . .	28
4.2.11	Slider . . . . .	28
4.2.12	Gauge . . . . .	28
4.2.13	Static text . . . . .	28
4.2.14	Check box . . . . .	28
4.2.15	Radio button . . . . .	28
4.2.16	Radio box . . . . .	29
4.2.17	Choice . . . . .	29
4.2.18	Combo Box . . . . .	29
4.2.19	List Box . . . . .	29

---

4.2.20	StaticLine . . . . .	29
4.2.21	Static bitmap . . . . .	29
4.2.22	List Control . . . . .	29
4.2.23	Tree Control . . . . .	29
4.2.24	Grid . . . . .	29
4.2.25	Custom Widget . . . . .	30
4.2.26	Spacer . . . . .	30
<b>5</b>	<b>Menu, Toolbar and Statusbar</b>	<b>31</b>
5.1	Introduction . . . . .	31
5.2	Menu . . . . .	31
5.3	Toolbar . . . . .	31
5.4	Statusbar . . . . .	31
<b>A</b>	<b>wxGlade License Agreement</b>	<b>32</b>
<b>B</b>	<b>Licenses and Acknowledgements for Incorporated Software</b>	<b>33</b>
B.1	OrderedDict . . . . .	33

---

# List of Figures

1.1	wxGlade windows . . . . .	1
3.1	The Main Palette . . . . .	11
3.2	The Tree Window . . . . .	12
3.3	The menu for a widget . . . . .	12
3.4	The menu for a sizer . . . . .	13
3.5	The Design Window . . . . .	13
3.6	Application Properties . . . . .	15
3.7	Common Properties . . . . .	16
3.8	Changing Common Properties . . . . .	17
3.9	Common Properties of a subclassed widget (default behaviour) . . . . .	18
3.10	Common Properties with Base class(es) entry . . . . .	18
3.11	Common Properties with a variable assignment . . . . .	19
3.12	Layout Properties . . . . .	20
3.13	Widget Properties . . . . .	21
3.14	Events Properties . . . . .	22
3.15	Events Properties with entered event handler name . . . . .	22
3.16	Properties for extra code and extra properties . . . . .	23
3.17	Set extra property . . . . .	24

# List of Examples

1.1	Installing wxGlade at /opt/wxglade . . . . .	4
1.2	Starting wxGlade at /opt/wxglade/bin/wxglade . . . . .	4
2.1	Correct entered wx constant . . . . .	6
2.2	Detailed application start code in Perl . . . . .	8
2.3	Simplified application start code in Perl . . . . .	9
2.4	Compiling a single file C++ project on Linux . . . . .	9
2.5	Compiling a multi file C++ project on Linux . . . . .	9
2.6	Converting a XRC file into a wxGlade project . . . . .	9
2.7	wxPython code to load and show a XRC resource . . . . .	10
3.1	Generated Python code of a subclassed widget . . . . .	18
3.2	Generated Python code of a widget with two base classes . . . . .	18
3.3	Generated Python code for a variable assignment . . . . .	19
3.4	Generated Python code of an <b>EVT_TEXT</b> event handler stub at line 12 . . . . .	22
3.5	Generated Python code for setting property <b>MaxLength</b> to <b>10</b> at line 14 . . . . .	24

# Preface

This manual describes Alberto Griggio's wxGlade program, a Python, Perl, Lisp, C++ and XRC Graphical User Interface ("GUI") editor for UNIX and Microsoft Windows. Each of the chapters in this manual is designed as a tutorial for using wxGlade and a reference for widgets supported until now.

## Contacts

Check the project homepage <http://wxglade.sourceforge.net> for the mailing list to discuss the project. Use the lists for questions, proposals, bug reports and collaboration. Information, support and bug reports can be addressed to the wxGlade mailing list too.

Any kind of feedback is always welcome.

Marcello Semboli [dinogen@siena.linux.it](mailto:dinogen@siena.linux.it) has written the original version of this document. The current version of this document is maintained by Carsten Grohmann.

## Copyrights and Trademarks

wxGlade is Copyright 2002-2007 by Alberto Griggio. Use and distribution of wxGlade is governed by the MIT license, located in Appendix A.

wxWidgets is Copyright (C) 1998-2005 Julian Smart, Robert Roebling et al. See: <http://www.wxwidgets.org> for details.

UNIX is a registered trademark of the X Open Group, Inc.

Microsoft and Windows are registered trademarks of Microsoft Corporation.

## Abbreviations

The following abbreviations are used in this manual:

**GUI** Graphical User Interface

**OS** Operating system

**SAE** Standalone Edition

**wx** abbreviation for wxWidgets

**wxg** File extension used by wxGlade to store the project in a XML file.

**wxWidgets** wxWidgets a widget toolkit and tools library for creating graphical user interfaces (GUIs) for cross-platform applications.

wxWidgets is open source and written in C++.

**WYSIWYG** What You See Is What You Get.

---

**X11** The X Window System version 11.

**XRC** XML-based system for describing wxWidgets resources like dialogs, menus or toolbars.

Those resources are loaded into the application at run-time.

**i18n** Numeronyms for internationalisation support.

Internationalisation means adapting software to different languages, regional differences, ...

**gettext** Widespread internationalisation (i18n) and localisation system.

---



## Chapter 1

# Introduction to wxGlade

### 1.1 What is wxGlade?

wxGlade is an open source graphical user interface builder written in Python using popular widget toolkit wxWidgets.

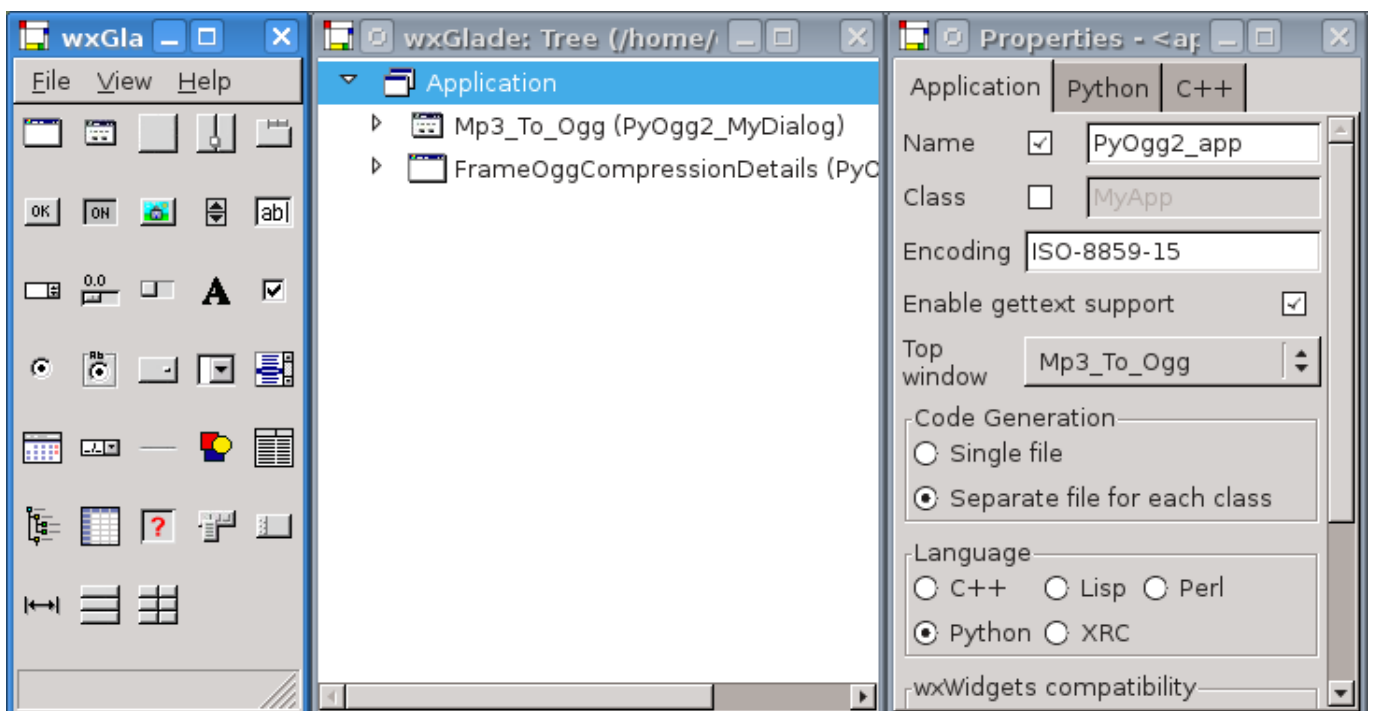


Figure 1.1: wxGlade windows

wxGlade allows to create graphical user interfaces using wxWidgets. The designer can arrange different widgets using a drag and drop WYSIWYG editor. This simplifies the creation of a graphical user interface in comparison with manual coded graphical user interfaces.

wxGlade is able to generate source code for Python, Perl, Lisp, C++ and XRC based on the designed GUI.

As you can guess by the name, its model is Glade, the famous GTK+/GNOME GUI builder, with which wxGlade shares the philosophy and the look & feel (but not a line of code).

## 1.2 What Can You Do with wxGlade?

With wxGlade you can:

- Design the whole GUI of your application inclusive simple or complex dialogs as well as menu bars, different kinds of buttons and text widgets, bitmaps, ...
- Use the graphical editor for editing, cutting and pasting widgets
- Convert your design in source code of your favorite language
- Run wxGlade on a wide variety of operation systems since it is written in Python

## 1.3 What is wxGlade NOT?

wxGlade is not a full featured IDE and will never be one. wxGlade is just a graphical user interface builder. The generated code does nothing apart from displaying the created widgets.

If you are looking for a complete IDE, maybe Boa Constructor <http://boa-creator.sourceforge.net> or PythonCard <http://www.pythoncard.com/> is the right tool.

## 1.4 Basics

You need to know the basics of wxWidgets or wxPython, as well as the basics of C++, Python, Perl or Lisp. You can't use wxGlade if you do not have any basic understanding of programming. You can't learn wxWidgets programming from reading this manual either.

## 1.5 Requirements and Supported Platforms

wxGlade has been tested and run on Windows, Linux, Mac OSX.

Because wxGlade is written in Python using wxPython, it can also be run on any platform that supports Python and wxPython. Especially the wxGlade requirements are:

- Python 2 - at least 2.3 or any later version of Python 2
- wxPython 2.6 or 2.8
- wxWidgets 2.6 or 2.8, the wxWidgets are often bundled with wxPython
- wxPython 2.9 is not supported yet

wxWidgets is available at <http://www.wxwidgets.org> and wxPython at <http://www.wxpython.org>.

## 1.6 Download

Source and binary packages for stable versions are available at <http://sourceforge.net/projects/wxglade>.

You can get the development version from [Bitbucket.org](https://bitbucket.org/agriggio/wxglade/overview) at <https://bitbucket.org/agriggio/wxglade/overview> using anonymous [Mercurial \(hg\)](#) access.

---

## 1.7 Installation

wxGlade is available in 4 different package types:

- the sources packages (.zip and .tar.gz)
- the full installer at Microsoft Windows (wxGlade-VERSION-setup.exe)
- the installer of the standalone edition at Microsoft Windows (wxGlade-SAE-VERSION-setup.exe)
- development version fetched with Mercurial or downloaded the current packaged development version from <https://bitbucket.org>

### 1.7.1 Installing at Microsoft Windows

The default installer requires a local installation Python and wxPython. The wxWidgets are bundled with wxPython on Microsoft Windows. Thereby you don't need to install wxWidgets separately.

There is no need to install additional packages for the standalone edition, because the standalone edition includes the required parts of Python, wxPython and wxWidgets.

The installation process is quite simple. Just download the installer file, execute it and follow the installer instructions.

### 1.7.2 Installing at Unix/Linux

The current Linux distributions provide wxGlade packages. Use the distribution specific install mechanism to install the wxGlade package and all dependencies.

You may install wxGlade from the source package if your distribution doesn't contain a proper package.

### 1.7.3 Installing from Source

The installation from scratch requires Python, wxPython and wxWidgets. Those three components have to be installed first. Maybe you could use already packaged versions of those components for your operating system. Otherwise read the installation documentation of the missing components and follow the instructions.

There are two ways for installing wxGlade from source - single or multi user installation.

Download a source package or a development package in a first step.

#### 1.7.3.1 Single user installation

Extract the downloaded package into a separate directory e.g. a subdirectory below user's home directory. Change in this directory and execute the **wxglade** file on Unix/Linux or **wxglade.pyw** on Microsoft Windows.

That's all. Installations below users home directory don't require administrative permissions.

#### 1.7.3.2 Multi user installation - variant 1

The first variant of a multi user installation is very similar to Section 1.7.3.1 except the installation directory. And probably you need administrative permissions. You could extract the wxGlade source package e.g. into **c:\program file\wxglade** on Microsoft Windows or into **/opt/wxglade** on Unix/Linux.

#### 1.7.3.3 Multi user installation - variant 2

Extract the downloaded package into a temporary directory. Change in this directory and execute the Python setup script using **python setup.py** in a terminal window.

---

**Example 1.1** Installing wxGlade at /opt/wxglade

```
# python setup.py install --prefix /opt/wxglade
running install
running build
running build_py
creating build
creating build/lib.linux-i686-2.7
creating build/lib.linux-i686-2.7/wxglade
creating build/lib.linux-i686-2.7/wxglade/widgets
creating build/lib.linux-i686-2.7/wxglade/widgets/combo_box
[...]
copying docs/html/ch04s23.html -> /opt/wxglade/share/doc/wxglade/doc/html
copying docs/html/ch04s26.html -> /opt/wxglade/share/doc/wxglade/doc/html
copying docs/html/ch05s02.html -> /opt/wxglade/share/doc/wxglade/doc/html
copying docs/html/pr01.html -> /opt/wxglade/share/doc/wxglade/doc/html
creating /opt/wxglade/share/doc/wxglade/doc/pdf
copying docs/pdf/manual.pdf -> /opt/wxglade/share/doc/wxglade/doc/pdf
creating /opt/share/man
creating /opt/share/man/man1
copying docs/man/wxglade.1 -> /opt/wxglade/share/man/man1
copying docs/man/manpage.xml -> /opt/wxglade/share/doc/wxglade
copying docs/src/manual.xml -> /opt/wxglade/share/doc/wxglade
running install_egg_info
Writing /opt/wxglade/lib/python2.7/site-packages/wxGlade-0.6.5-py2.7.egg-info
```

After the installation has finished the wxGlade main script **wxglade** is located at **<install directory>/bin**.

Execute the script to start wxGlade

**Example 1.2** Starting wxGlade at /opt/wxglade/bin/wxglade

```
# /opt/wxglade/bin/wxglade
Starting wxGlade version 0.6.5 on Python 2.7.2+
Base directory:           /opt/wxglade/lib/python2.7/site-packages/wxglade
Documentation directory:   /opt/wxglade/lib/python2.7/site-packages/wxglade/docs
Icons directory:          /opt/wxglade/lib/python2.7/site-packages/wxglade/icons
Build-in widgets directory: /opt/wxglade/lib/python2.7/site-packages/wxglade/widgets
Template directory:        /opt/wxglade/lib/python2.7/site-packages/wxglade/templates
Credits file:              /opt/wxglade/share/doc/wxglade/credits.txt
License file:              /opt/wxglade/share/doc/wxglade/license.txt
Tutorial file:             /opt/wxglade/lib/python2.7/site-packages/wxglade/docs/html/ ↵
                             index.html
Using wxPython 2.8.12.1
loaded code generator for perl
loaded code generator for XRC
loaded code generator for python
loaded code generator for lisp
loaded code generator for C++
Found widgets listing -> /opt/wxglade/lib/python2.7/site-packages/wxglade/widgets/widgets. ↵
                             txt
loading widget modules:
    frame
    dialog
[...]
```

## Chapter 2

# Exploring wxGlade

### 2.1 Quick start

We will design a simple form.

Start wxGlade by running the **wxglade** program on Unix platforms or the **wxglade.pyw** program on Microsoft Windows.

You will see a Main Palette with several buttons, and a Tree Window with an icon marked Application. A Properties Window shows the properties of the Application.

If you move the mouse over a button in the main window, a tooltip will display its function.

To add a frame in the design window, from the Main Palette choose the first button: “Add a frame”.

Then choose `wxFrame` as the base class.

Look at the tree window and see that two icons are generated under the application icon, a frame icon and a sizer icon.

If you double click with the mouse on the frame icon, the designer window appears. Notice that the sizer is displayed as a set of gray boxes: they are the “slots” of the grid sizer where you will place the widgets.

You put a widget on a sizer by selecting it on the Main Window, then click on an empty slot on the frame on the designer window. Try adding a static text, a text control and a button.

If you want to add something else, add empty slots on the sizer by right-clicking on the sizer on the tree window and selecting “Add slot”.

Play around, adding four or five widgets on the frame.

Now look at the properties form; there are three tabs. In the “Common” tab you can specify the name, size and color of the widget.

In the “Layout” tab you can adjust borders and alignments.

In the “Widget” tab you find the properties depending on the widget.

You can select the properties of a widget by clicking on the designer window or the corresponding icon on the tree window.

Try adjusting widgets with the properties form until you know you have played enough.

Now let’s generate the code.

Select the Application icon on the tree window and go to the properties window.

Check Name and Class, choose a “Top window”, check “Single file” and choose the language and set the “Output path” by pushing the button for selecting a path and a filename.

Finally press the “Generate code” button, and the code is generated.

Compile and enjoy.

---

## 2.2 Basics of wxGlade

The program wxGlade is a tool for designing Graphical User Interfaces (GUI). It is intended to be used with the wxWidgets framework in all its flavors: C++, Lisp, Perl, Python and XRC.

You use a visual editor for creating forms, menus and toolbars with the mouse.

Your design is saved in a `.wxg` file, which is the wxGlade file format. Then you generate source code or XRC by using visual tools or invoking wxGlade at the command line. You can also use wxGlade in your makefile by generating source code only when the `.wxg` file changes.

A `.wxg` file can contain multiple forms, panels, menus and toolbars and generate either a single file containing all classes or multiple files containing one class each.

wxGlade does not manage events, file inclusion, function names, stubs or anything else but graphic interface code.

## 2.3 Best Practice

The main goal of the recommendations is to improve the usability and maintainability of code generated by wxGlade. The recommendations combine the experience of many wxGlade users.

**Always overwrite existing sources** wxGlade is able to adapt existing source files after the design has changed. This feature have some limitations e.g. in case of name changes and changed dependencies. Thereby it's recommended to overwrite existing sources always and extend derived classes with your functionality.

**Use new namespace for Python projects** It's generally recommended to use the new namespace ("**import wx**") for wxPython projects. The old one ("**from wxPython.wx import \***") has some significant drawbacks like potential namespace conflicts.

**Use the C++ naming convention** Use the C++ names for all wx identifies like classes, colours or events of the wx framework. Please don't enter identifiers already formatted in a language specific form. wxGlade is able to transform the entered original identifiers in language-specific terms. You can use your own style for your object certainly.

---

**Example 2.1** Correct entered wx constant

---

Enter "**wxID\_CANCEL**" even for wxPython instead of "**wx.ID\_CANCEL**"

---

**Always use UTF-8 encoding** It's generally recommended to use Unicode encoding for all non-ASCII character sets.

**Always use gettext support** Enable internationalisation support. There are no disadvantages if internationalization is active but not used.

It's hard to add i18n and Unicode afterwards from project point of view.

**Suggestion on naming** The wxWidgets are written in C++ and follow the C++ naming convention. This naming convention may differ from the language specific and / or project specific naming convention.

For consistency's sake, it's recommended to use the wxWidgets style.

## 2.4 Command line invocation

You can run wxGlade without parameters to start the GUI on an empty application as follows:

**wxglade**

Run wxGlade GUI on an existing application specifying the `.wxg` file as follow:

**wxglade <WVG File>**

If you only want to generate the code without starting the GUI, use the `-g` or `--generate-code` option with the language as argument as follows:

---

**wxglade -g <LANGUAGE> <WXG File>**

**wxglade --generate-code=<LANGUAGE> <WXG File>**

Possible values for LANGUAGE are "XRC", "python", "perl", "lisp" or "C++".

You can also specify the destination of the generated code with `-o` or `--output` option:

**wxglade -g <LANGUAGE> -o <DESTINATION> <WXG File>**

The DESTINATION argument can be a file or a directory. If DESTINATION is a file, wxGlade will generate single-file source code. In case DESTINATION is a directory wxGlade will generate multiple-file source code.

This is the complete description of the command line:

```
# wxglade --help
Usage: wxglade <WXG File>          start the wxGlade GUI
or:   wxglade <Options> <WXG File> generate code from command line
or:   wxglade --version             show programs version number and exit
or:   wxglade -h|--help             show this help message and exit
Options:
  --version                show program's version number and exit
  -h, --help               show this help message and exit
  -g LANG, --generate-code=LANG
                           (required) output language, valid languages are: C++,
                           XRC, lisp, perl, python
  -o PATH, --output=PATH
                           (optional) output file in single-file mode or output
                           directory in multi-file mode
Example: Generate Python code out of myapp.wxg
  wxglade -o temp -g python myapp.wxg
Report bugs to: <wxglade-general@lists.sourceforge.net> or at
                <http://sourceforge.net/projects/wxglade/>
wxGlade home page: <http://wxglade.sourceforge.net/>
```

---

#### Note

Use **wxglade.pyw** instead of **wxglade** on Microsoft Windows.

---

## 2.5 Using the source code

There are a lot of options to control the source code generation process. They are bundled in the “Properties” window (see Figure 3.6). Let’s talk about three of those options - “Single file”, “Separate file for each class” and “Overwrite existing sources”.

The first two options triggers wxGlade to generate one file with all classes inside or multiple files - one per class/widget. The “Single file” option includes source and header file for C++ certainly.

The third option “Overwrite existing sources” is just about control - “Full control by wxGlade” and “Shared control”. It separated the two ways to work with wxGlade.

### 2.5.1 Full control by wxGlade

If “Overwrite existing sources” is set, wxGlade will re-generated all source files and drop potential manual changes. You’ve to include the generated source files and use derived classes for implementing changes.

The files written by wxGlade are consistent always. Also if e.g. classes or attributes are renamed. Rewriting the whole files is less error-prone in comparison with Section 2.5.2. That is the advantages of this method.

This method is the recommended one.

---

### 2.5.2 Shared control

Manual changes in the source files won't be overwritten if "Overwrite existing sources" isn't set. You can safely edit the source code of the generated class. This is because wxGlade marks the untouchable code with the special comments "**begin wxGlade**" and "**end wxGlade**". So you can edit all you need outside these two tags. When you make changes in your forms, a new code generation will not modify the user code. wxGlade is applying most of the changes but not all changes. Especially renamed classes and attributes need additional attention.

---

#### Note

Overwriting multiple files is not recommended as well as overwriting of files with percent character ("%") inside is not supported.

---

### 2.5.3 Output path and filenames

"Output path" specifies the name of the output file for "Single file" projects or the output directory for multi-file projects ("Separate file for each class"). The filename has to include the appropriate suffix of the programming language always. An exception is the "Output path" for "Single file" C++ projects. Filename don't contains the filename extension now. The extension for C++ source and header files will be appended later automatically.

### 2.5.4 Automatically created wxApp instance

wxGlade is able to extent the created source for by a code sequence to create and start an instance of projects "Top window".

In case "Name" and "Class" are set, a detailed start code with a derived class of wxApp will be created. If just "Name" is given, a simplified start code will be generated.

There is a short explanation of "Class" and "Name" in Section [3.4.1](#).

The application start code of a multi-file project will be recreated every time the code generation is running.

In opposition the application start code of single-file projects will not updated if the name of the "Top window" has changed and "Overwrite existing sources" is not set.

---

#### Example 2.2 Detailed application start code in Perl

---

```
package MyApp;
use base qw(Wx::App);
use strict;
sub OnInit {
    my( $self ) = shift;
    Wx::InitAllImageHandlers();
    my $frame_1 = MyFrame->new();
    $self->SetTopWindow($frame_1);
    $frame_1->Show(1);
    return 1;
}
# end of class MyApp
package main;
unless (caller) {
    my $local = Wx::Locale->new("English", "en", "en"); # replace with ??
    $local->AddCatalog("app"); # replace with the appropriate catalog name
    my $app = MyApp->new();
    $app->MainLoop();
}
```

---



**Example 2.3** Simplified application start code in Perl

```
package main;
unless (caller) {
    my $local = Wx::Locale->new("English", "en", "en"); # replace with ??
    $local->AddCatalog("PlOgg1_app"); # replace with the appropriate catalog name
    local *Wx::App::OnInit = sub{1};
    my $PlOgg1_app = Wx::App->new();
    Wx::InitAllImageHandlers();
    my $Mp3_To_Ogg = PlOgg1_MyDialog->new();
    $PlOgg1_app->SetTopWindow($Mp3_To_Ogg);
    $Mp3_To_Ogg->Show(1);
    $PlOgg1_app->MainLoop();
}
```

**2.5.5 Compiling C++ code**

You can compile your wxGlade project after the generation of the C++ source and header files. The following examples demonstrate compiling on Linux command line using **g++**.

**Example 2.4** Compiling a single file C++ project on Linux

```
# g++ FontColour.cpp $(wx-config --libs) $(wx-config --cxxflags) -o FontColour
# ll FontColour*
-rwxr-xr-x 1 carsten carsten 72493 Jun 15 09:22 FontColour
-rwxr-xr-x 1 carsten carsten 1785 Mai 11 19:24 FontColour.cpp
-rwxr-xr-x 1 carsten carsten 1089 Jun 11 07:09 FontColour.h
```

**Example 2.5** Compiling a multi file C++ project on Linux

```
# g++ CPPOgg2_main.cpp $(wx-config --libs) $(wx-config --cxxflags) \
    -o CPPOgg2_main CPPOgg2_MyDialog.cpp CPPOgg2_MyFrame.cpp
# ll CPPOgg2*
-rwxr-xr-x 1 carsten carsten 108354 Jun 15 09:33 CPPOgg2_main
-rwxr-xr-x 1 carsten carsten 844 Mai 11 19:25 CPPOgg2_main.cpp
-rw-r--r-- 1 carsten carsten 5287 Mai 18 19:06 CPPOgg2_MyDialog.cpp
-rw-r--r-- 1 carsten carsten 1829 Jun 11 07:11 CPPOgg2_MyDialog.h
-rw-r--r-- 1 carsten carsten 1785 Mai 11 19:25 CPPOgg2_MyFrame.cpp
-rw-r--r-- 1 carsten carsten 1290 Jun 11 07:10 CPPOgg2_MyFrame.h
```

**2.6 Handling XRC files**

wxGlade is able to save projects as XRC files and to convert XRC files into wxGlade projects.

One way for converting XRC files is the usage of the Python script **xrc2wxg.py** at command line. The script is part of wxGlade.

**Example 2.6** Converting a XRC file into a wxGlade project

```
# ./xrc2wxg.py FontColour.xrc

# ls -l FontColour.*
-rw-r--r-- 1 carsten carsten 5554 Dez 4 20:36 FontColour.wxg
-rw-r--r-- 1 carsten carsten 4992 Dez 4 20:13 FontColour.xrc
```

The “File” menu provides a menu item “Import from XRC...” to import and open a XRC file directly.

The following example shows how to load and show the frame “Main” from XRC file `test.xrc`.

---

**Example 2.7** wxPython code to load and show a XRC resource

---

```
#!/usr/bin/env python

import wx
from wx import xrc

GUI_FILENAME = "test.xrc"
GUI_MAINFRAME_NAME = "Main"

class MyApp(wx.App):
    def OnInit(self):
        self.res = xrc.XmlResource(GUI_FILENAME)
        self.frame = self.res.LoadFrame(None, GUI_MAINFRAME_NAME)
        self.frame.Show()
        return True

if __name__ == '__main__':
    app = MyApp()
    app.MainLoop()
```

---

## 2.7 Specifying the path of bitmaps

In wxGlade some widgets need to specify a bitmap path. You can use any graphic format supported by wxWidgets.

The bitmap can be specified in several ways:

Usually you can type an absolute path in a text box or browse for a bitmap with a file dialog. This will produce a `wxBitmap` object with the typed string as bitmap path (e.g. `wxBitmap("/usr/share/icons/application.png", wxBITMAP_TYPE_ANY)`)

You can enter a variable name using the `var:` tag in the text box. This will produce a `wxBitmap` object with the variable name as bitmap path (e.g. `var:my_bitmap_path` produces `wxBitmap(my_bitmap_path, wxBITMAP_TYPE_ANY)`). In Perl code generation a “\$” sign is added if you omit it.

You can enter a code chunk returning a `wxBitmap`, by using the `code:` tag. This inserts verbatim the code you enter in brackets and nothing more (e.g.: if `wxSomeWidget` needs a `wxBitmap` as an argument, the string `code:if (x == 0) get_bitmap1() else get_bitmap2();` produces `wxSomeWidget((if (x == 0) get_bitmap1() else get_bitmap2());, option1, option2)`).

wxGlade never declares or assigns variable or function names, so after code generation, you have to provide extra code to declare your variables or functions.

If you use `var:` or `code:` tags the preview window shows an empty bitmap of fixed size.

---

## Chapter 3

# wxGlade User Interface

### 3.1 Main Palette

The main window is a palette that hosts the menu and the widget choice buttons.

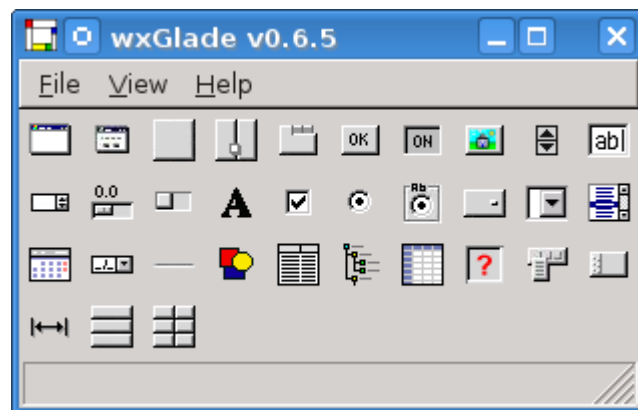


Figure 3.1: The Main Palette

If you pass the mouse pointer over a button a tooltip shows the button's description.

The “Add a Frame” button and the “Add a Dialog/Panel” button bring up a dialog to add a frame, a dialog or a panel to your project.

The “Add a MenuBar” button asks you for the name of the class then adds a menu bar to your project.

The “Add a ToolBar” button asks you for the name of the class then adds a toolbar to your project.

The other buttons in the main window add widgets to a form. When you click on one, the mouse pointer changes to an arrow. Then you can click on a sizer's empty cell to add the widget to it.

### 3.2 Tree Window

The tree window shows the logical hierarchy of widgets and its child-widgets. For example you can see a panel as a tree's node and the widgets on it as child nodes.

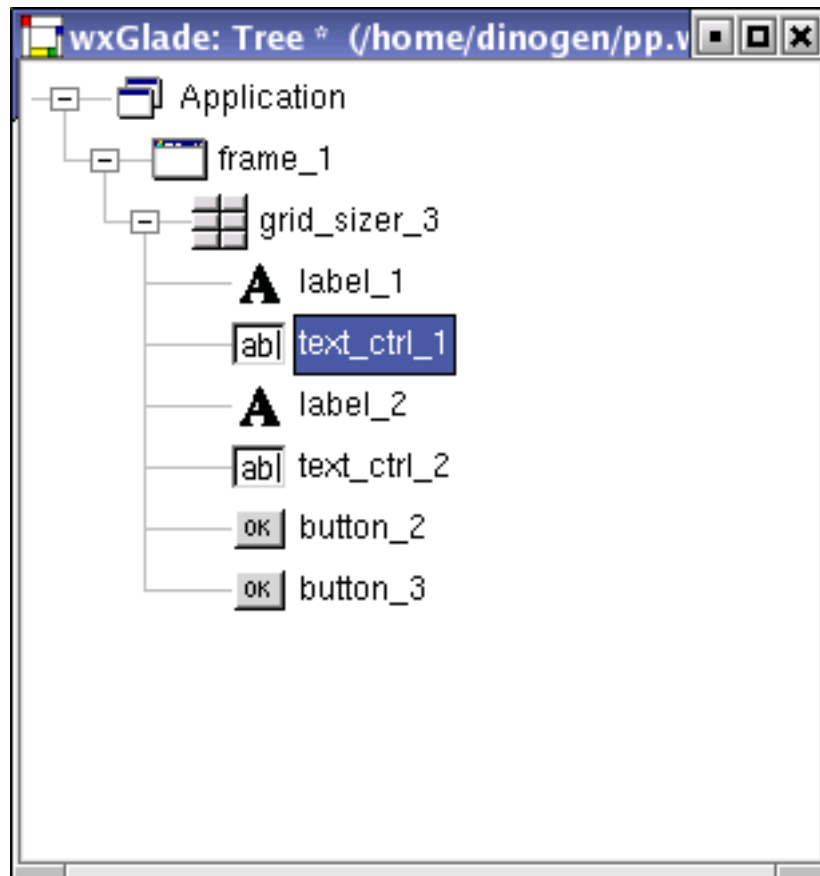


Figure 3.2: The Tree Window

You can show or hide the tree window by the menu item View/Show Tree.

Usually a frame or a panel contains a sizer, so you often see a sort of panel-sizer-widgets structure. The tree gets more complex when you nest sizers within sizers.

You can navigate the visual presentation of your widget tree by mouse, expand and collapse sizers, and copy, cut or remove widgets.

A click on an icon in the tree window displays the properties of the corresponding element in the properties window. A double click in a frame, dialog or panel icon makes the designer window show it as it appears. Clicking with the right button of the mouse gives you a pop-up menu.

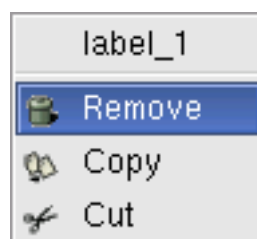


Figure 3.3: The menu for a widget

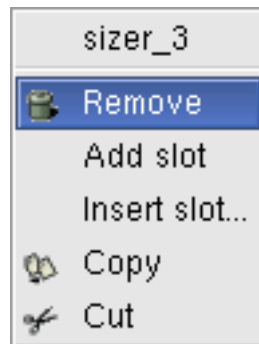


Figure 3.4: The menu for a sizer

The pop-up menu for a widget allows you to copy, cut or remove the element. The pop-up menu for a sizer allows you to copy, cut or remove the element, or add or insert an empty slot.

---

**Note**

Often when you add an empty slot, you have to make the designer window larger, to show the new slot.

---

### 3.3 Design Window

The design window shows the frame or panel you are creating in WYSIWYG mode and allows you to select a widget from the main palette and to put it on an empty slot of a sizer. You can show the design window by double-clicking on the icon of a frame or dialog in the tree window.

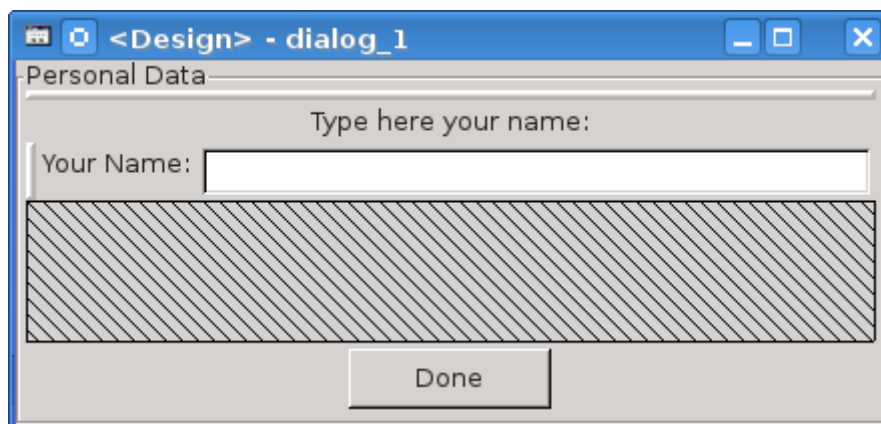


Figure 3.5: The Design Window

By clicking with the right mouse button on a widget you can access the context menu. Notice that the sizers, which are invisible elements, have a little gray “handle,” that you can click to select the sizer or let the pop-up menu appear.

The pop-up menu is the same as the one you get in the Tree Window, as shown in Figure 3.3 or in Figure 3.4.

### 3.4 Properties Window

The properties window lets you see and edit the properties that apply to the selected element. This window consists up to six different tabs. All six tabs are not always present. The visibility of the single tabs depends on the widget type. Most widgets

---

have a “Common” tab and a “Code” tab. The combination of presented tabs depends on the widget type.

For example:

- `wxFrame` widgets have “Common”, “Widget” and “Code” tabs
- `Spacers` have the tabs “Layout” and “Code”
- `wxGridSizer` widgets have “Common” and “Grid”
- `wxBoxSizer` widgets only have the “Common” tab

Editing properties is quite simple; Properties are represented by buttons, text boxes, checks and other controls. Usually they are referenced by the same name or symbol that you find writing C++ code.

Usually you get the changes in the design window in real time. In some cases you have to push the “Apply” button. For example, the `wxNotebook` widget shows in its properties window a list of child `wxPanels`. You have to press the “Apply” button to show changes you make when you add or remove panels.

You can show or hide the properties window by the menu item View → Show Properties.

### 3.4.1 Common Properties

The first tab contains the common properties that apply to all widgets. As shown in Figure 3.7 the common properties are related to name, class, size, colors, fonts and tooltip.

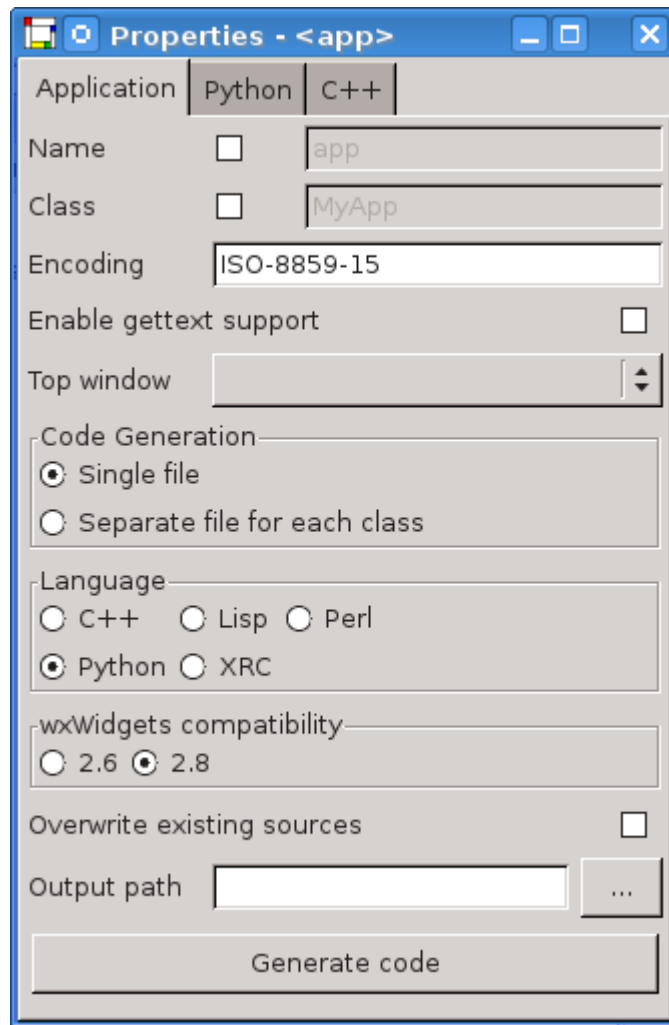


Figure 3.6: Application Properties

**“Name”**

Name of the instance created from “Class”

Section 2.5.4 provides more information

**“Class”**

Name of the automatically generated class derived from `wxApp`

Section 2.5.4 provides more information

**“Encoding”**

Encoding of the generated source files.

The encoding to use with new projects will be determined automatically based on the machine settings. “UTF-8” will be used if the automatic detection fails.

**“Enable gettext support”**

Enable internationalisation and localisation for the generated source files

**“Top window”**

This widget is used as top window in the `wxApp` start code

Section 2.5.4 provides more information

**“Code Generation”**

Write all source code in one file or split the source into one file per class / widget

Section 2.5 provides more information

**“Language”**

Programming language to generate the source files in

**“wxWidgets compatibility”**

Generate source files for the selected wxWidgets version

**“Overwrite existing sources”**

Overwrite existing source files or modify the code sequences generated by wxGlade in place

Section 2.5 provides more information

**“Output path”**

Output file or directory

Section 2.5.3 provides more information

**“Generate code”**

Start generating source files

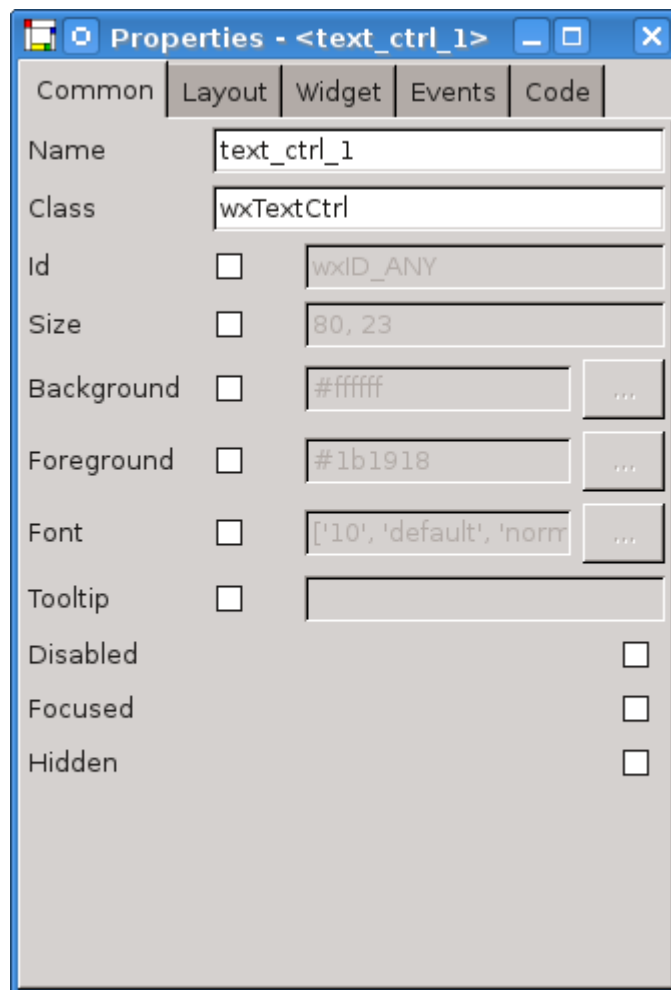


Figure 3.7: Common Properties



The property name is a mangled version of the wxWidgets property name. The property input field is disabled by default. wxGlade won't use disabled properties for code generation. wxWidgets defaults are used instead.

Enable the property in the wxGlade GUI to set non-default values (see Figure 3.8).

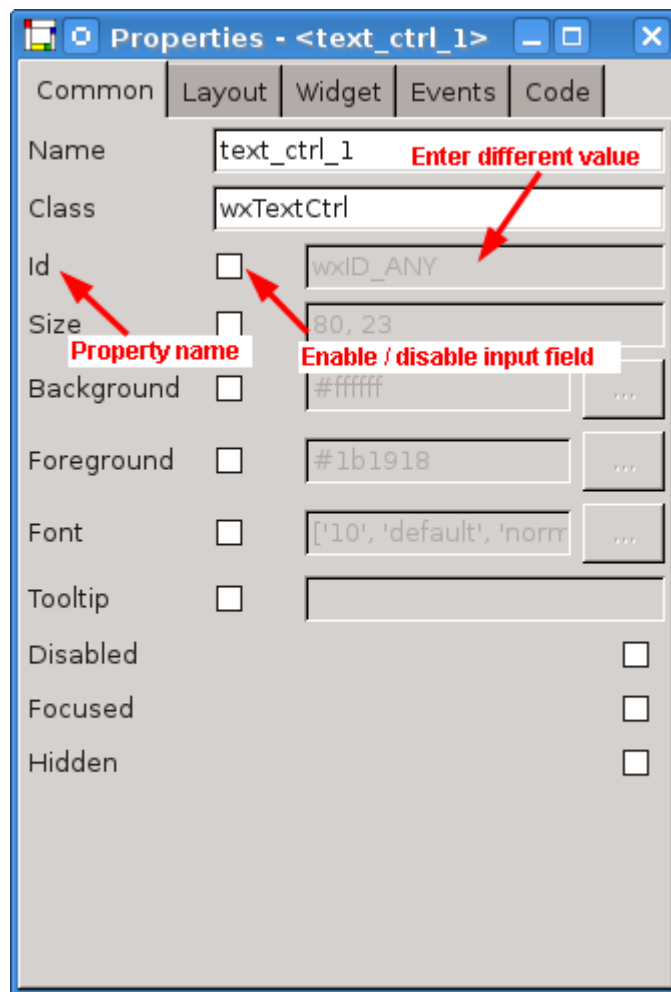


Figure 3.8: Changing Common Properties

#### “Name”

Name of the instance created from “Class”

#### “Class”

Name of the subclass of the widget. How this name affects code generation depends on the output language.

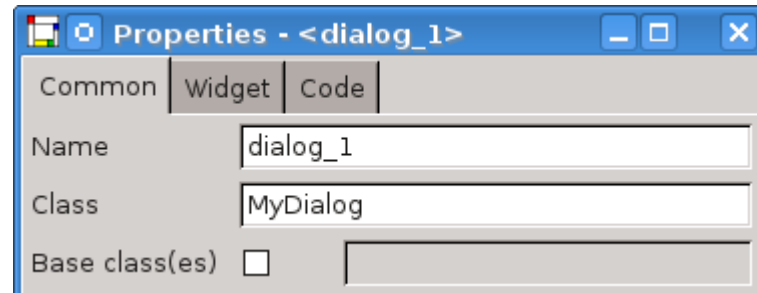


Figure 3.9: Common Properties of a subclassed widget (default behaviour)

**Example 3.1** Generated Python code of a subclassed widget

```

1 class MyDialog(wxDialog):
2     def __init__(self, *args, **kwargs):
3         # begin wxGlade: MyDialog.__init__
4         kwargs["style"] = wx.DEFAULT_DIALOG_STYLE
5         wxDialog.__init__(self, *args, **kwargs)

```

**“Base class(es)”**

A comma-separated list of custom base classes. The first will be invoked with the same parameters as this class, while for the others the default constructor will be used. This property will be shown only for non-managed widgets for instance `wxFrame`, `wxDialog`, `wxNotebook`, `wxPanel` and `wxSplitterWindow`. You should probably not use this if “overwrite existing sources” is not set.

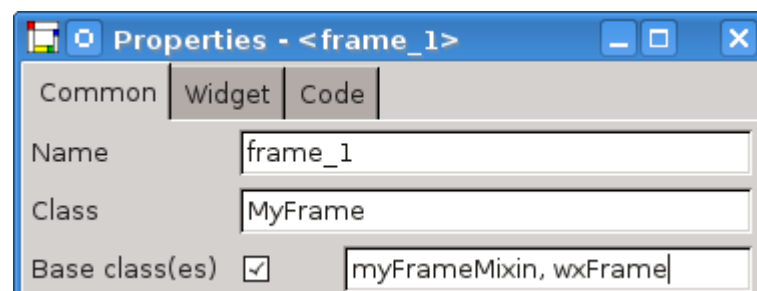


Figure 3.10: Common Properties with Base class(es) entry

**Example 3.2** Generated Python code of a widget with two base classes

```

1 class MyFrame(myFrameMixin, wxFrame):
2     def __init__(self, *args, **kwargs):
3         # begin wxGlade: MyFrame.__init__
4         kwargs["style"] = wx.DEFAULT_FRAME_STYLE
5         myFrameMixin.__init__(self, *args, **kwargs)
6         wxFrame.__init__(self)

```

**“Id”**

This property could be

- a constant numeric value
- a predefined identifier e.g. `wxID_ANY`
- a predefined variable like a class member e.g. `self.myButtonID`

- a variable assignment e.g. “**self.myButtonID=?**” The pattern of a variable assignment is always “**variable=value**”. The value could be again a numeric value, a predefined identifier, another predefined variable or “?” a shortcut for “**wxNewId()**”

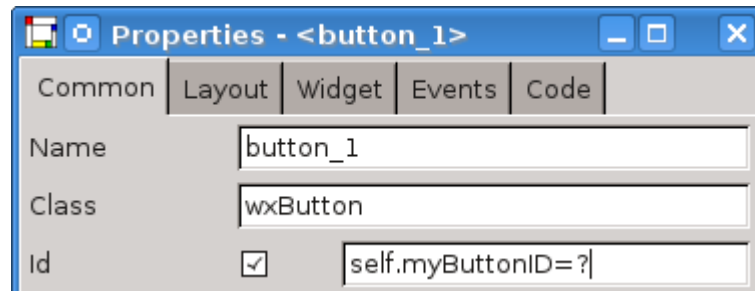


Figure 3.11: Common Properties with a variable assignment

---

### Example 3.3 Generated Python code for a variable assignment

---

```
class MyFrame(wx.Frame):
    def __init__(self, *args, **kwargs):
        # begin wxGlade: MyFrame.__init__
        kwargs["style"] = wx.DEFAULT_FRAME_STYLE
        wx.Frame.__init__(self, *args, **kwargs)
        self.myButtonID = wx.NewId()
        self.button_1 = wx.Button(self, self.myButtonID, "button_1")
        self.__set_properties()
        self.__do_layout()
        # end wxGlade
```

---

#### “Size”

Set the widget size in pixels.

#### “Background”

Set the background colour of the widget.

#### “Foreground”

Set the foreground colour of the widget.

#### “Font”

Set the font for widgets text elements.

#### “Tooltip”

Set a tooltip for this widget.

#### “Disabled”

Disable the widget.

#### “Focused”

Sets the widget to receive keyboard input.

#### “Hidden”

Hide the widget.

---

### 3.4.2 Layout Properties

The second tab is related to layout properties that control position and resizing within the sizer.

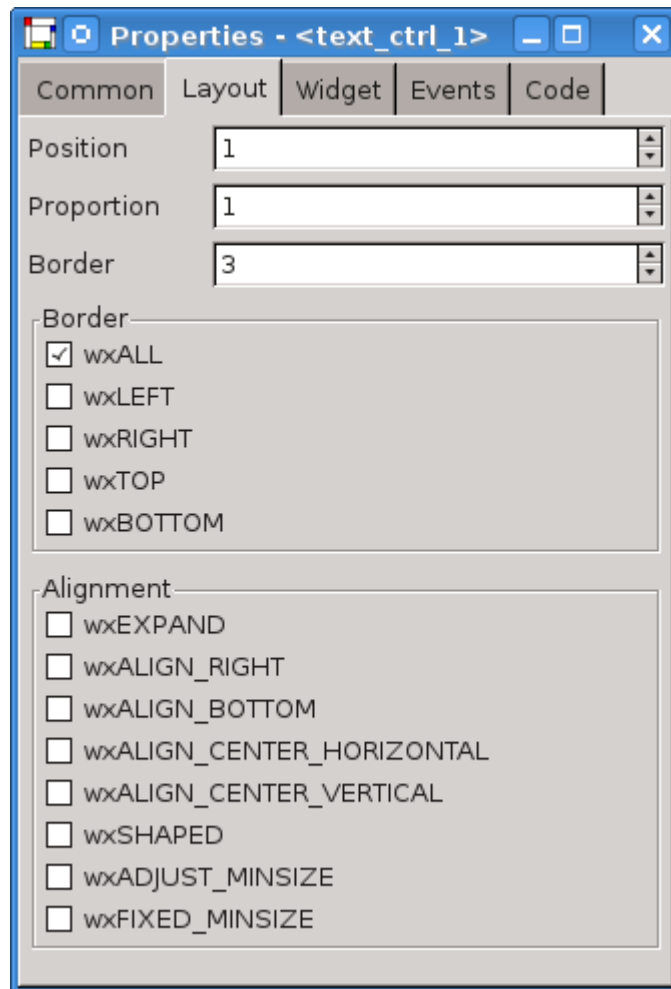


Figure 3.12: Layout Properties

These properties apply to any widget. You can check or uncheck any option related to the placement in the sizer. Many widgets may have a default value of 3 in the “Border” property in the Preferences Dialog (see Section 3.5). If you let a widget have a default border, the `wxALL` option is also checked.

### 3.4.3 Widget Properties

The third tab, named “Widget” is different for each widget, and lets you edit properties for the specific element you have selected.

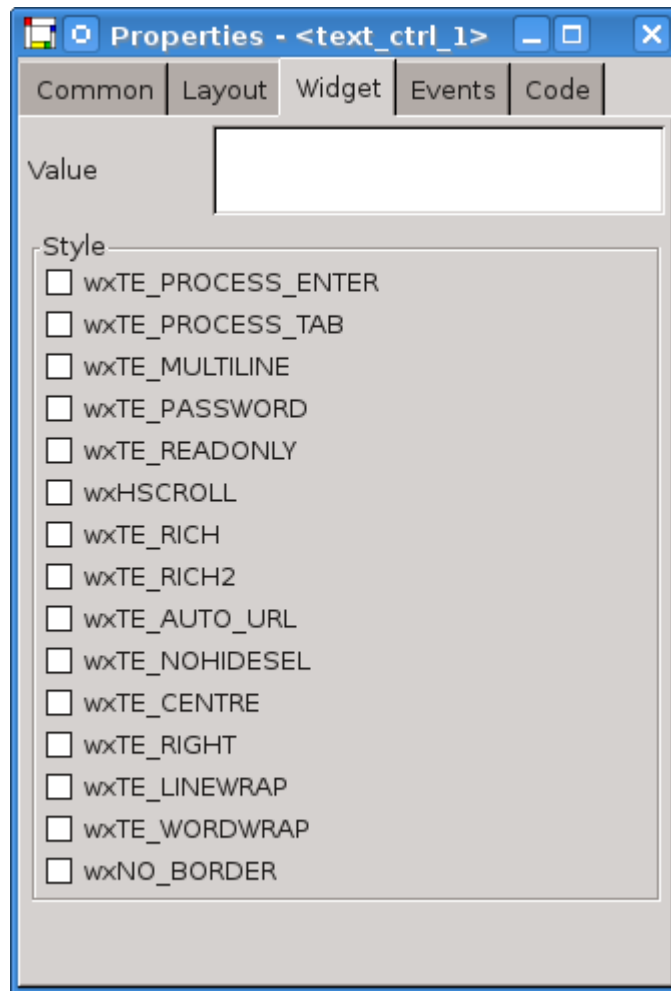


Figure 3.13: Widget Properties

The set of options may also be quite complex in the case of widgets that have a great deal of methods and properties (such as grids and tree views). In this case, wxGlade greatly simplifies the process of designing forms.

### 3.4.4 Events Properties

The fourth tab, named “Events” lists the widgets events. wxGlade generates an event handler stub and binds the event for each added handler name.

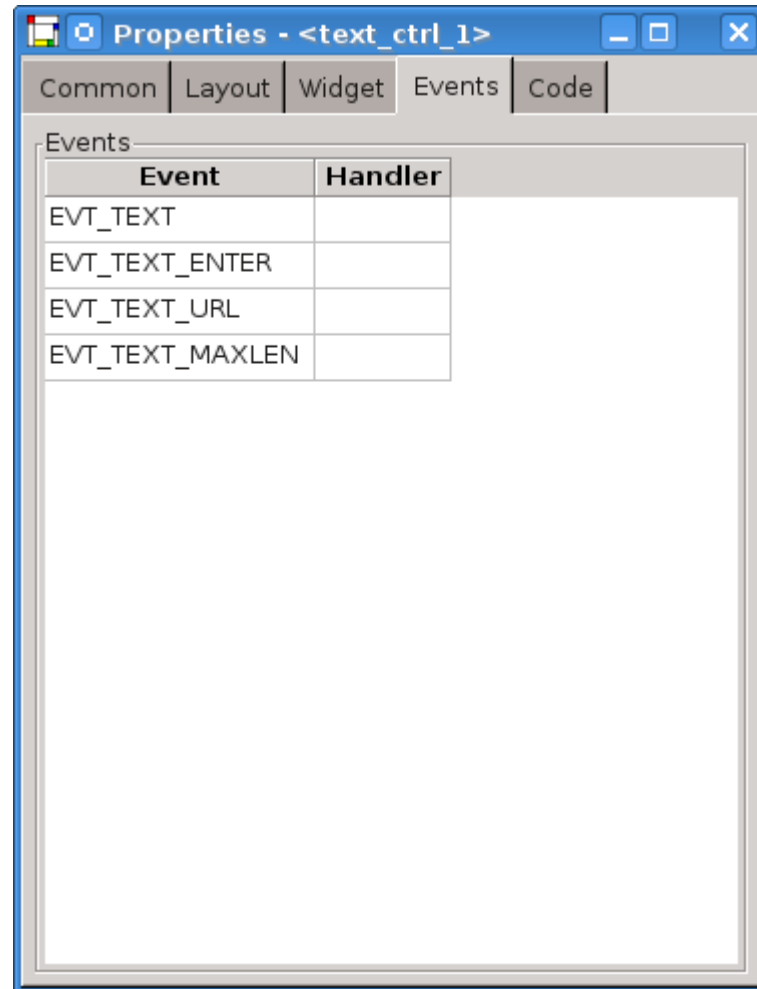


Figure 3.14: Events Properties

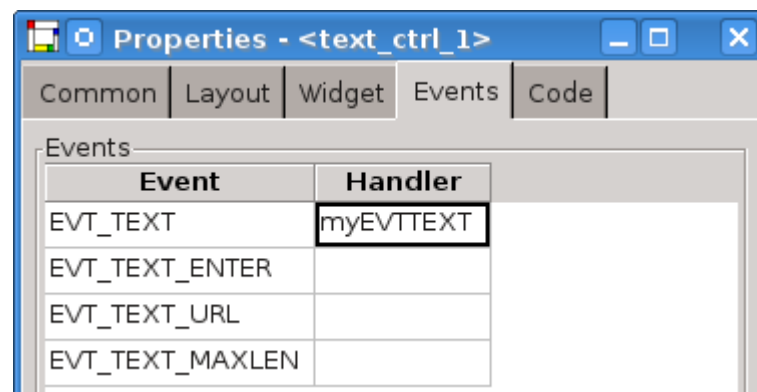


Figure 3.15: Events Properties with entered event handler name

---

**Example 3.4** Generated Python code of an **EVT\_TEXT** event handler stub at line 12

---

```
1 class MyFrame(wx.Frame):  
2     def __init__(self, *args, **kwargs):  
3         # begin wxGlade: MyFrame.__init__
```

```

4      kwds["style"] = wx.DEFAULT_FRAME_STYLE
5      wx.Frame.__init__(self, *args, **kwds)
6      self.text_ctrl_1 = wx.TextCtrl(self, -1, "")
7      self.__set_properties()
8      self.__do_layout()
9      self.Bind(wx.EVT_TEXT, self.myEVTTEXT, self.text_ctrl_1)
10     # end wxGlade
11 def myEVTTEXT(self, event): # wxGlade: MyFrame.<event_handler>
12     print "Event handler 'myEVTTEXT' not implemented!"
13     event.Skip()

```

### 3.4.5 Code Properties

The fifth and last tab is named “Code” and has two parts.

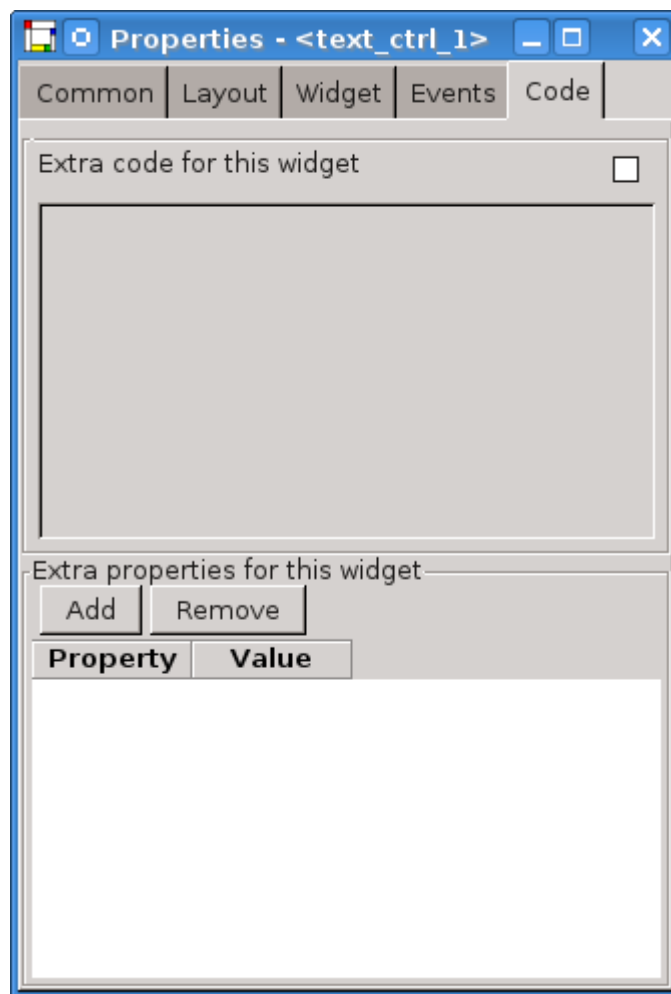


Figure 3.16: Properties for extra code and extra properties

The upper part provides the ability to add additional code for that widget e.g. for importing a custom class. This “Extra code” will be added to the context of the source file and not to the context of the class.

The under part simplifies setting of additional widget properties. Add the property name to the “Property” field and not the name of the setter function. For instance add “**MaxLength**” and not “**SetMaxLength**”. The “Value” field is just a text field. You can

enter e.g. a simple number only as well as a complex statement e.g. `0, 0, "1"` or a function call. But be carefully! Your entered sequence will be inserted in the source without any changes - one to one.

---

#### Note

“Extra code” and “Extra properties” won’t be processed for the widget preview.

---

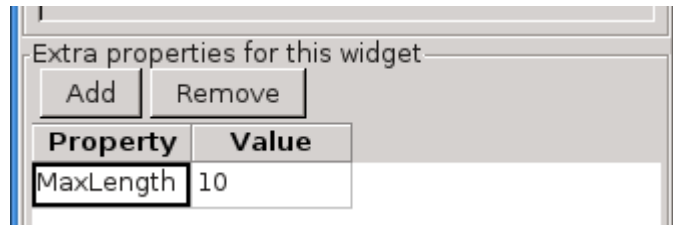


Figure 3.17: Set extra property

---

#### Example 3.5 Generated Python code for setting property **MaxLength** to **10** at line 14

---

```

1 class MyFrame(wx.Frame):
2     def __init__(self, *args, **kwargs):
3         # begin wxGlade: MyFrame.__init__
4         kwargs["style"] = wx.DEFAULT_FRAME_STYLE
5         wx.Frame.__init__(self, *args, **kwargs)
6         self.text_ctrl_1 = wx.TextCtrl(self, -1, "")
7         self.__set_properties()
8         self.__do_layout()
9         # end wxGlade
10    def __set_properties(self):
11        # begin wxGlade: MyFrame.__set_properties
12        self.SetTitle("frame_1")
13        self.text_ctrl_1.SetMaxLength(10)
14        # end wxGlade

```

---

## 3.5 Preferences Dialog

You can access the Preferences Dialog with the menu item View → Preferences. You can choose some decoration options, like whether to show icons in menus or not, but also something more effective. For example, you can modify the number of buttons in the Main Palette. If you type a value of 15 or 30, you get a long toolbar-like Main Palette. You can also choose the default path where you save wxGlade files or generate source code.

Another useful option is to enable a default border of 3 around some widgets. In many cases this can be useful to have set.

You need to restart wxGlade for changes to take effect.

## 3.6 The wxGlade Menu

wxGlade has only a few very small menus.

---



### 3.6.1 The FILE menu

In the FILE menu there are the classic File → New, File → Open... and File → Save items. When opening or saving a new file, the file dialog defaults to the directory that you put in the “Initial path” textbox in the Preferences dialog, usually the user home directory.

The File → Generate code item produces the code from the current design.

### 3.6.2 The VIEW menu

In the VIEW menu, you can show or hide the tree window and the properties window.

In this menu you access the Preferences Dialog as well.

### 3.6.3 The HELP menu

The HELP menu provides access to the wxGlade user manual (this documentation) as well as to the “About...” dialog.

## 3.7 Shortcuts

#### Ctrl-G

Generate code from the current GUI design

#### Ctrl-I

Import GUI design out of a XRC file

#### Ctrl-N

Start a new GUI design

#### Ctrl-O

Read a GUI design from a .wxg file

#### Ctrl-S

Save the current GUI design to a .wxg file

#### Shift-Ctrl-S

Save the current GUI design to another .wxg file

#### Ctrl-P

Open a preview window for the current top-level widget

#### Ctrl-Q

Exit wxGlade

#### Ctrl-C

Copy the selected item, element, text, ...

#### Ctrl-V

Insert clipboard content

#### Ctrl-X

Cut the selected item, element, text, ...

---

**F1**

Show the wxGlade user manual (this documentation)

**F2**

Show the Tree window

**F3**

Show the Properties window

**F4**

Show all application windows

**F5**

Refresh the screen

---

## Chapter 4

# Supported widgets

### 4.1 Introduction

wxGlade supports a number of widgets and helps you to edit the properties and visual look of each one.

### 4.2 Widget list

Follow the widget list as it appears in the wxGlade main window.

#### 4.2.1 Frame

This prompts for a `wxFrame` or a `wxMDIChildFrame`. A vertical `wxBoxSizer` is appended. In the properties window you can choose the styles and you can add an icon.

#### 4.2.2 Dialog or Panel

This prompts for a `wxDialog` or a `wxPanel` in top level. In the properties window you can choose the styles and, for the dialog, you can add an icon.

#### 4.2.3 Panel

This allows you to add a panel to a sizer.

In the properties window you can choose the styles.

#### 4.2.4 Splitter window

This produces a `wxSplitterWindow` and two associated panels as well. You can choose vertical or horizontal splitting.

In the properties window you can choose the styles and the sash position.

Be careful not to put too large a widget in a splitter panel, because while it might appear normal in the design window, when you run your program one of two panels will take all the available space and the other will shrink to the minimum size possible.

### 4.2.5 Notebook

This produces a `wxNotebook` and one panel for each tab.

In the properties window you can add and remove tabs, which appear in a list.

Don't forget to click on the "Apply" button to transfer changes that you have made in the list to the design window.

### 4.2.6 Button

This produces a `wxButton`. You can enter a caption and the "default" flag. If you want to add an image you need a bitmap button (see Section 4.2.8).

### 4.2.7 Toggle button

This produces a `wxToggleButton`. You can enter a caption and the status (clicked or not) of the button.

### 4.2.8 Bitmap button

This produces a `wxBitmapButton`. You can set the "default" flag on or off. You also can choose the bitmap for the button and, optionally, the bitmap for the disabled status. Refer to Section 2.7 for bitmap path specifications.

### 4.2.9 Text control

This produces a `wxTextCtrl`. In the properties window you can enter the text and also set the style.

### 4.2.10 Spin control

This produces a `wxSpinCtrl`. In the properties window you can enter the value, the range and also set the style.

### 4.2.11 Slider

This produces a `wxSlider`. In the properties window you can enter the value, the range and also set the style.

### 4.2.12 Gauge

This produces a `wxGauge`. In the properties window you can enter the range and set the style.

### 4.2.13 Static text

This produces a `wxStaticText`. In the properties window you can enter the text, set the style and tell wxGlade whether to store the control as an attribute.

### 4.2.14 Check box

This produces a `wxCheckBox`. In the properties window you can enter the text, and the status, checked or not, of the button.

### 4.2.15 Radio button

This produces a `wxRadioButton`. In the properties window you can enter the text, and the status, clicked or not, and the style.

---

#### 4.2.16 Radio box

This produces a `wxRadioBox`. In the properties window you can enter the dimension. The style determines whether the dimension is the number of rows or columns.

You also can set which button is selected with the “Selection” spin starting from 0. You can edit the list of choices, but remember to click on the “Apply” button to consolidate changes.

#### 4.2.17 Choice

This produces a `wxChoice`. In the properties window you can enter the position of the selected item starting from 0. You can edit the list of choices, but remember to click on the “Apply” button to consolidate changes.

#### 4.2.18 Combo Box

This produces a `wxComboBox`. In the properties window you can enter the position of the selected item starting from 0. You can edit the list of choices, but remember to click on the “Apply” button to consolidate changes.

#### 4.2.19 List Box

This produces a `wxListBox`. In the properties window you can enter the position of the selected item starting from 0. You can edit the list of choices, but remember to click on the “Apply” button to consolidate changes.

#### 4.2.20 StaticLine

This produces a vertical or horizontal `wxStaticLine`. In the properties window you can tell wxGlade whether to store the object as an attribute of the frame class.

#### 4.2.21 Static bitmap

This produces a `wxStaticBitmap`. You will be prompted for the bitmap path. Refer to Section 2.7 for bitmap path specifications. In the properties window you can set the style and you can tell wxGlade whether to store the object as an attribute of the frame class.

#### 4.2.22 List Control

This produces a `wxListCtrl`. In the properties window you can set the style.

#### 4.2.23 Tree Control

This produces a `wxTreeCtrl`. In the properties window you can set the style.

#### 4.2.24 Grid

This produces a `wxGrid`. In the properties window you can set the style, the row number, the label size, the line and background color and the selection mode. You can edit the list of columns, but remember to click on the “Apply” button to consolidate changes. Also you can choose to let wxGlade to create the grid or leave it to the user code.

---

#### 4.2.25 Custom Widget

When you put a custom widget in the design window you will be prompted for a class name. In the properties window you can set a number of custom attributes that will appear in the constructor call. These attributes have different effects in C++, Lisp, Perl, Python or XRC code generation. Four special attributes `$id`, `$parent`, `$width` and `$height` return the value you specify in the “Common” tab of the custom widget.

#### 4.2.26 Spacer

When you put a spacer into a sizer slot in the design window you will be prompted for the size; wxGlade will generate the code to set an empty space in that slot of the sizer.

## Chapter 5

# Menu, Toolbar and Statusbar

### 5.1 Introduction

wxGlade helps you to design the menu and the toolbar for your application.

You can create the menu and toolbar as stand alone classes by clicking the corresponding button in the main window.

Alternatively you can make the menu, toolbar and statusbar associated with a `wxFrame`, by selecting the related checkboxes in the `wxFrame` properties window.

### 5.2 Menu

In the menu properties window click on the “Edit menus...” button. A dialog will let you edit your menu. Use the “Add” button to add items to the menu; enter the label, an optional name and help string. You can use numbers or variable names as the item id. If you use a variable name, you have to provide extra code in the generated source code.

Choose the type of the item: Normal, Checkable or Radio.

You can move menu items with “Up” and “Down” buttons, and you can modify the hierarchy of the menu with “<” and “>” buttons.

### 5.3 Toolbar

You can edit the toolbar’s style and bitmap size in the properties window.

Click on the “Edit tools...” button to edit the toolbar buttons. Use the “Add” button to add buttons to the toolbar; enter the label, an optional name and help string. You can use numbers or variable names as the button id. If you use a variable name, you have to provide extra code in the generated source code.

Choose the type of the button: Normal, Checkable or Radio.

You can move toolbar buttons with “Up” and “Down” buttons.

You have to enter two bitmaps, one for normal status and the other for the pushed status.

Refer to Section [2.7](#) for bitmap path specifications.

### 5.4 Statusbar

In the properties window you can edit the list of fields and their size, but remember to click on the “Apply” button to consolidate changes.

---

## Appendix A

# wxGlade License Agreement

Copyright (c) 2002-2007 Alberto Griggio [agriggio@users.sourceforge.net](mailto:agriggio@users.sourceforge.net)

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.



## Appendix B

# Licenses and Acknowledgements for Incorporated Software

This section lists licenses and acknowledgements for third-party software incorporated in wxGlade.

### B.1 OrderedDict

The `OrderedDict` class version 1.1 has been integrated. The class is downloaded from <http://pypi.python.org/pypi/ordereddict> and contains following notice:

Copyright (c) 2009 Raymond Hettinger

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

---