

# wxGlade manual

---

# Contents

<b>Preface</b>	<b>viii</b>
<b>1 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 Requirements and Supported Platforms . . . . .	2
1.5 Installation . . . . .	2
1.5.1 Download . . . . .	3
1.5.2 Installing at Microsoft Windows . . . . .	3
1.5.3 Installing at Unix/Unix-like Operating Systems . . . . .	3
1.5.4 Installing from Source . . . . .	3
1.6 Configuring wxGlade . . . . .	5
1.6.1 Preferences Dialog . . . . .	5
1.6.2 Configuration files and Configuration directory . . . . .	5
1.6.3 Environment Variables . . . . .	5
1.6.4 Logging . . . . .	6
1.7 How to Report a Bug . . . . .	6
1.8 Deprecated features . . . . .	7
<b>2 Exploring wxGlade</b>	<b>8</b>
2.1 Quick example . . . . .	8
2.2 Basics of wxGlade . . . . .	9
2.3 Escape Sequences . . . . .	9
2.4 Best Practice . . . . .	9
2.5 Language specific peculiarities . . . . .	10
2.5.1 Python . . . . .	10
2.5.2 Lisp . . . . .	10
2.6 Command line invocation . . . . .	10
2.7 Using the source code . . . . .	11
2.7.1 Full control by wxGlade . . . . .	11

---

---

2.7.2	Shared control	11
2.7.3	Output path and filenames	12
2.7.4	Automatically created wxApp instance	12
2.7.5	Compiling C++ code	13
2.8	Handling XRC files	13
<b>3</b>	<b>wxGlade User Interface</b>	<b>15</b>
3.1	Main Palette	15
3.2	Tree Window	15
3.3	Design Window	17
3.4	Properties Window	18
3.4.1	Application Properties	18
3.4.2	Common Properties	22
3.4.3	Layout Properties	25
3.4.4	Widget Properties	26
3.4.5	Events Properties	27
3.4.6	Code Properties	29
3.5	The wxGlade Menu	30
3.5.1	The FILE menu	30
3.5.2	The VIEW menu	30
3.5.3	The HELP menu	31
3.6	Shortcuts	31
<b>4</b>	<b>Supported widgets</b>	<b>32</b>
4.1	Introduction	32
4.2	Specifying the path of bitmaps	32
4.3	Widget List	32
4.3.1	Frame	32
4.3.2	Dialog or Panel	32
4.3.3	Panel	33
4.3.4	Splitter Window	33
4.3.5	Notebook	33
4.3.6	Buttons	33
4.3.7	Gauge	33
4.3.8	Hyperlink Control	33
4.3.9	Radio Box	34
4.3.10	Spin Control	34
4.3.11	Slider	34
4.3.12	Static Text	34

---

---

4.3.13	Text Control	34
4.3.14	Check Box	34
4.3.15	Choice	34
4.3.16	Combo Box	34
4.3.17	List Box	34
4.3.18	Static Line	34
4.3.19	Static Bitmap	35
4.3.20	List Control	35
4.3.21	Tree Control	35
4.3.22	Grid	35
4.3.23	Custom Widget	35
4.3.24	Spacer	35
<b>5</b>	<b>Menu, Statusbar and Toolbar</b>	<b>36</b>
5.1	Introduction	36
5.2	Menu	36
5.3	Statusbar	37
5.4	Toolbar	37
<b>6</b>	<b>wxGlade technical notes</b>	<b>39</b>
6.1	Startup	39
6.2	Adding a toplevel widget	39
6.3	Adding a toplevel sizer	40
6.4	Adding a normal widget/sizer	40
6.5	Changing the value of a Property	40
6.6	Saving the app	41
6.7	Loading an app from a XML file	41
6.8	Generating the source code	41
6.9	For contributors	41
<b>7</b>	<b>Installing and Designing own Widget Plugins</b>	<b>43</b>
7.1	Widgets packages	43
7.1.1	Create ZIP package	44
7.2	Installing Widget Plugins locally	44
7.3	Designing own Widget Plugins	44
7.3.1	Widget Initialisation	45
<b>A</b>	<b>Abbreviations</b>	<b>46</b>
<b>B</b>	<b>Copyrights and Trademarks</b>	<b>47</b>
<b>C</b>	<b>wxGlade License Agreement</b>	<b>48</b>
<b>D</b>	<b>Licenses and Acknowledgements for Incorporated Software</b>	<b>49</b>
D.1	OrderedDict	49

---

# List of Figures

1.1	wxGlade windows . . . . .	1
1.2	wxGlade preferences dialog . . . . .	5
1.3	An error dialog example . . . . .	7
3.1	The Main Palette . . . . .	15
3.2	The Tree Window . . . . .	16
3.3	The menu for a widget . . . . .	16
3.4	The menu for a sizer . . . . .	17
3.5	The Design Window . . . . .	17
3.6	Project Properties - Application settings . . . . .	19
3.7	Project Properties - Language settings . . . . .	21
3.8	Common Properties . . . . .	22
3.9	Changing Common Properties . . . . .	23
3.10	Common Properties of a subclassed widget (default behaviour) . . . . .	23
3.11	Common Properties with Base class(es) entry . . . . .	24
3.12	Common Properties with a variable assignment . . . . .	24
3.13	Layout Properties . . . . .	26
3.14	Widget Properties . . . . .	27
3.15	Events Properties . . . . .	28
3.16	Events Properties with entered event handler name . . . . .	28
3.17	Properties for extra code and extra properties . . . . .	29
3.18	Set extra property . . . . .	30
5.1	Menu editor . . . . .	36
5.2	Statusbar properties . . . . .	37
5.3	Toolbar editor . . . . .	38

---

# List of Tables

2.1 Interaction between properties to generate different types of start code . . . . . 12

# 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 . . . . .	9
2.2	Detailed application start code in Perl . . . . .	12
2.3	Simplified application start code in Perl . . . . .	13
2.4	Compiling a single file C++ project on Linux . . . . .	13
2.5	Compiling a multi file C++ project on Linux . . . . .	13
2.6	Converting a XRC file into a wxGlade project . . . . .	14
2.7	wxPython code to load and show a XRC resource . . . . .	14
3.1	Generated Python code of a subclassed widget . . . . .	23
3.2	Generated Python code of a widget with two base classes . . . . .	24
3.3	Generated Python code for a variable assignment . . . . .	24
3.4	Generated Python code of an <b>EVT_TEXT</b> event handler stub at line 12 . . . . .	28
3.5	Generated Python code for setting property <b>MaxLength</b> to <b>10</b> at line 14 . . . . .	30
7.1	Directory package . . . . .	43
7.2	ZIP package . . . . .	43

# Preface

This manual describes the program wxGlade, initially written by Alberto Griggio. wxGlade is 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.

## Contact

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.

## License

wxGlade is copyright 2002-2007 by Alberto Griggio and 2011-2014 by Carsten Grohmann.

Use and distribution of wxGlade is governed by the MIT license, located in Appendix C, [wxGlade License Agreement](#).

---

## 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.

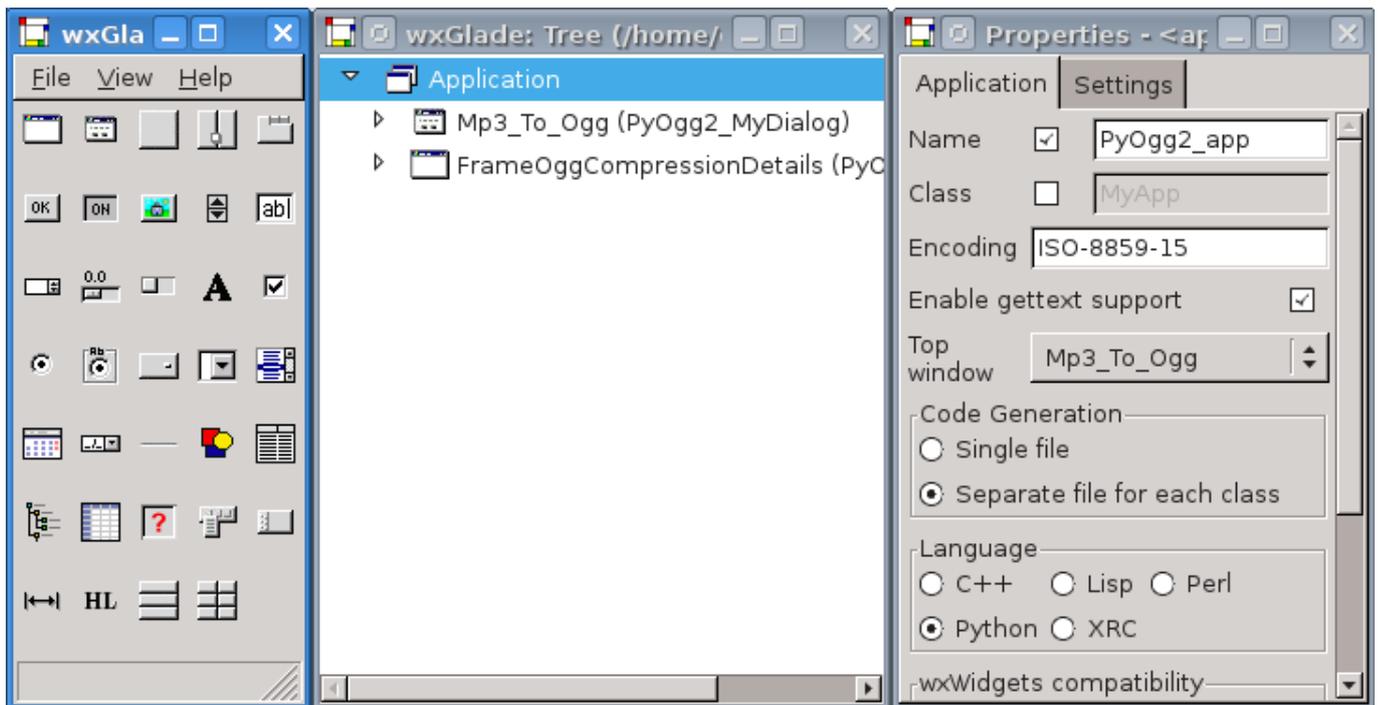


Figure 1.1: wxGlade windows

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](#) or [PythonCard](#) is the right tool.

wxGlade isn't a tool to learn programming with wxWidgets. You can't use wxGlade if you do not have any basic understanding of programming. You need to know the basics of wxWidgets, as well as the basics of C++, Python, Perl or Lisp.

## 1.4 Requirements and Supported Platforms

wxGlade has been tested and run on Microsoft Windows, Linux, OS X.

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 2.4 or any later version of Python 2
- wxPython 2.8 or 3.0  
Sometimes the wxPython module "wxversion" is packaged separately e.g. in Debian. Please install the "wxversion" package manually in such case,
- wxWidgets 2.8 or 3.0, the wxWidgets are often bundled with wxPython

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

## 1.5 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>

### 1.5.1 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.

### 1.5.2 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.

### 1.5.3 Installing at Unix/Unix-like Operating Systems

Current Linux distributions provide wxGlade packages already. 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 provide any package or the package is out-of-date.

### 1.5.4 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.

#### 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 operating systems or **wxglade.pyw** on Microsoft Windows.

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

#### Multi user installation - variant 1

The first variant of a multi user installation is very similar to Section 1.5.4, "[Single user installation](#)" 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.

#### 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
[...]
```

## 1.6 Configuring wxGlade

### 1.6.1 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.

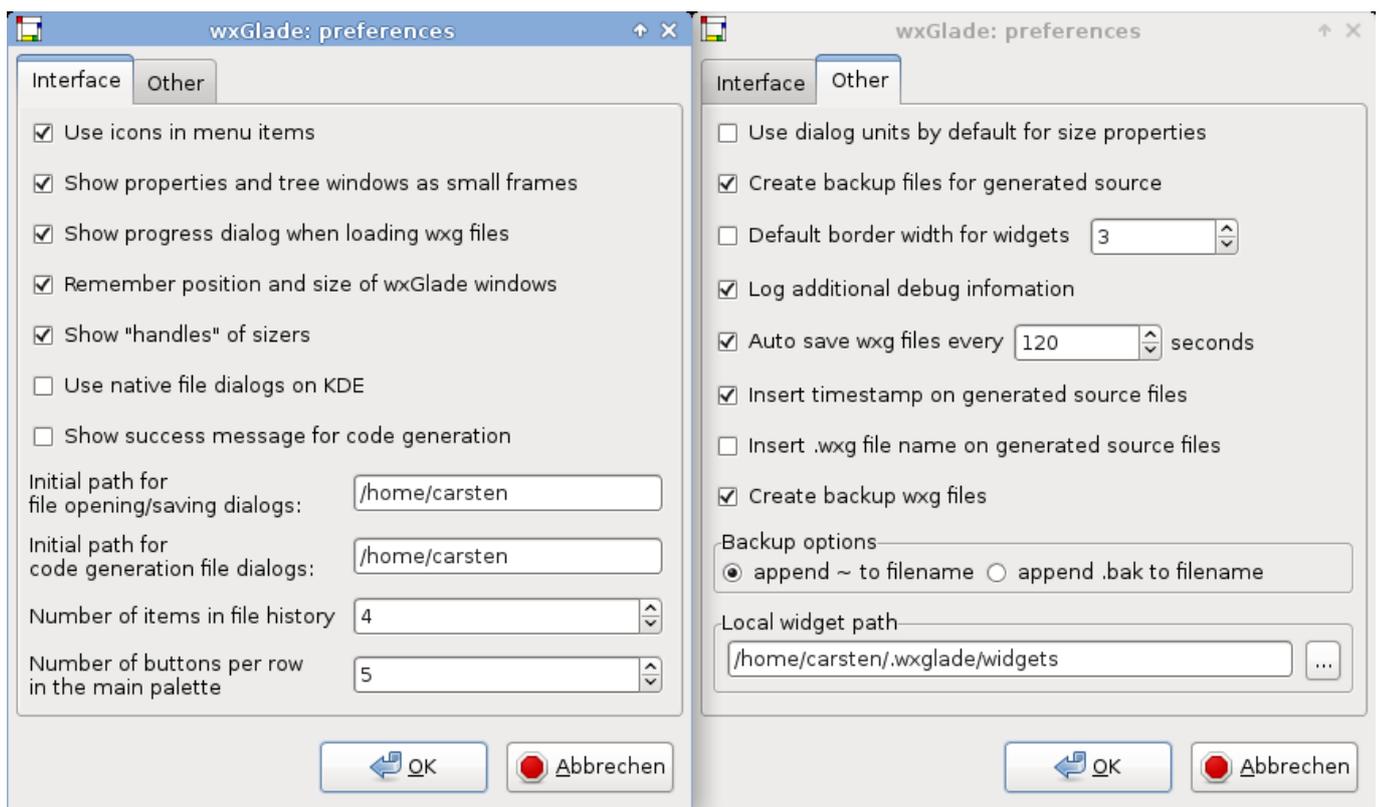


Figure 1.2: wxGlade preferences dialog

### 1.6.2 Configuration files and Configuration directory

wxGlade stores user related information in a own directory. The directory is named “~/ .wxglade” on Unix or in “Application Data/wxglade” on Microsoft Windows. The name of the directory “Application Data” will be translated to users language by Windows automatically.

The configuration file `wxgladerc` on Unix or `wxglade.ini` on Microsoft Windows contains users configuration. It’s a text file. Be carefully if you are changing the file manually. The last open files are listed in `file_history.txt`. The subdirectories templates contains wxGlade templates generated by the user itself.

The directory contains the log files also. The log files are detailed described in Section 1.6.4, “Logging”.

### 1.6.3 Environment Variables

wxGlade supportes only one environment variable “WXGLADE\_CONFIG\_PATH”.

If you want to store the whole configuration data inclusive user generated templates and log files in a non-default directory, then store the full path of the alternative directory in the environment variable “WXGLADE\_CONFIG\_PATH” and start wxGlade with the new environment.

### 1.6.4 Logging

wxGlade writes always a small error log file `wxglade.log`. The file size is limited to 100 KB. wxGlade will keep at most two log files `wxglade.log` and `wxglade.log.1`. The roll over from `wxglade.log` to `wxglade.log.1` will occur if the file size limit is reached. An already existing file `wxglade.log.1` will be deleted automatically. The log file is a text file in Unicode.

The log file location is described in Section 1.6.2, “[Configuration files and Configuration directory](#)”.

## 1.7 How to Report a Bug

Writing a helpful bug report is easy if you follow some hints. The items below should help you to integrate useful information. They are not an absolute rule - it’s more like a guideline.

- What did you? May you want to include a screenshot.
- What do you want to happen?
- What actually happened?
- Provide a short example to reproduce the issue.
- Include the internal error log file “`wxglade.log`” always.

May you read [How to Report Bugs Effectively](#) additionally.

Please open a new bug in the [wxGlade bug tracker](#) at Sourceforge. Alternatively you can send the bug report to the wxGlade mailing list. Keep in mind that you need a subscription for sending emails to this mailing list.

---

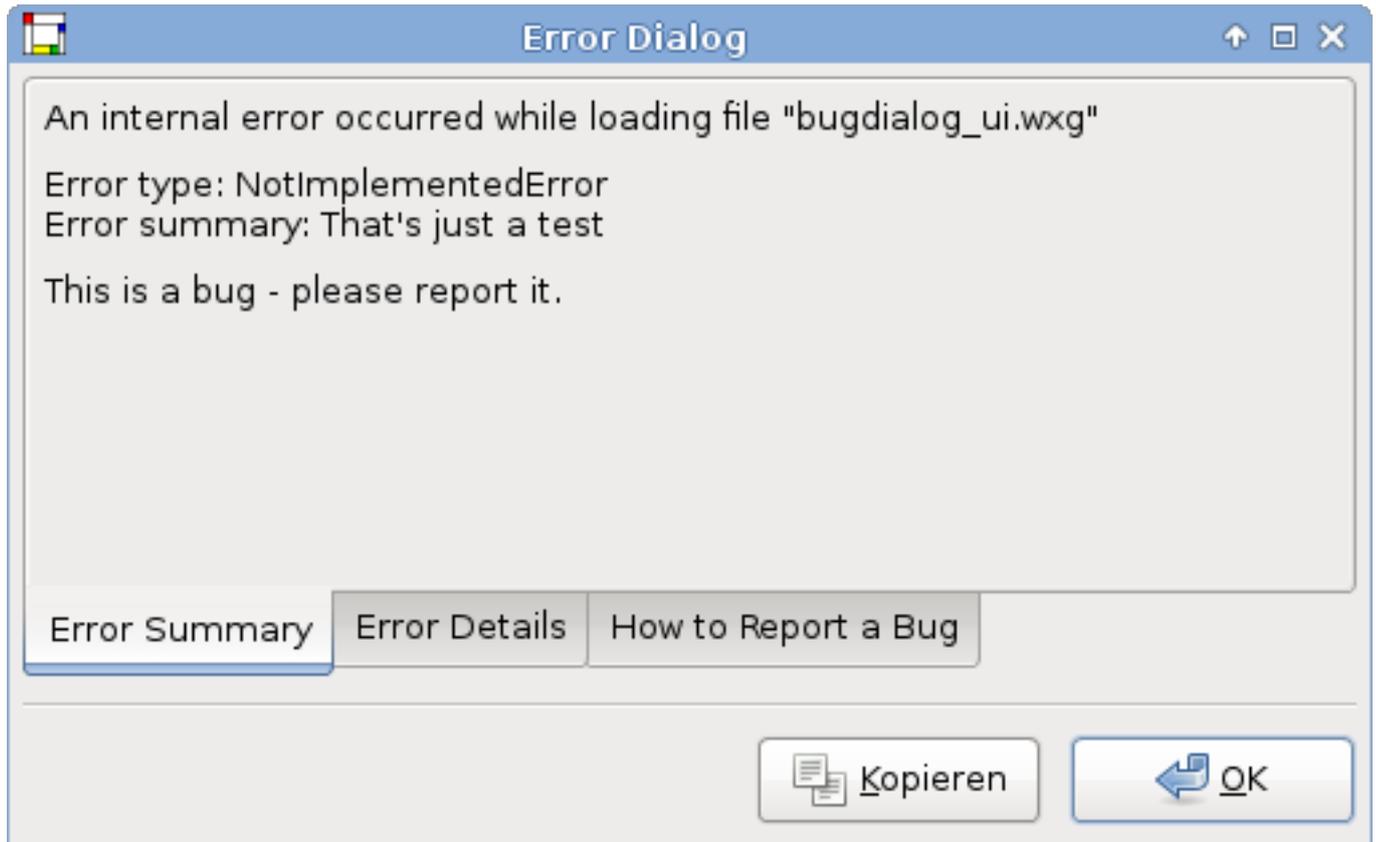


Figure 1.3: An error dialog example

## 1.8 Deprecated features

wxGlade can change code inside existing source files to reflect changed designs. This feature is deprecated now and will be removed within the next releases.

## Chapter 2

# Exploring wxGlade

### 2.1 Quick example

We will design a simple form.

Start wxGlade by running the **wxglade** program on Unix 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 Escape Sequences

Escape sequences are used to define certain special characters within string literals. wxGlade supports escape sequences generally. The only exception is the null byte (“`0`”) and the escape sequence (“`\0`”) belonging to it. wxGlade can’t handle null bytes.

Escape sequences like “`\n`” or “`\t`” will not be touched by wxGlade. Thereby the generated source code contains exactly the same sequence as entered. The language interpreter or compiler will interpret and probably convert the sequence into control characters. For example “`\n`” will be converted into a line break.

Escape sequences with at least two leading backslashes e.g. “`\\n`” will be escaped to show exactly the same sequence and don’t convert it into control characters. Question marks especially double quotes will be escaped also.

## 2.4 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 can change code inside existing source files to reflect changed designs. This feature has 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.

---

**Note**

This feature is deprecated now and will be removed within the next releases.

---

**Use the C++ naming convention** Use the C++ names for all wx identifiers 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.

**Prevent language specific statements** Usage of language specific codes e.g. for "Extra code for this widget" or in generic input fields complicated changing the output language later e.g. to re-use GUI elements in another project too.

## 2.5 Language specific peculiarities

### 2.5.1 Python

It's not recommended to use nested classed and functions in combination with disabled feature "Overwrite existing sources". Use derived classes to implement your functionality. See Section 2.4, "Best Practice" also.

### 2.5.2 Lisp

The Lisp code generated by wxGlade may or may not working with a current Lisp dialect. Help to improve the Lisp support is really welcome.

Unsupported features in Lisp:

- Unicode support
- Support for wxWidgets 3.0

## 2.6 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 <WXG 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.

---

## 2.7 Using the source code

There are a lot of options to control the source code generation process. They are bundled in the “Application” page of the “Properties” window (see Figure 3.6, “[Project Properties - Application settings](#)”). 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.

### 2.7.1 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 2.7.2, “[Shared control](#)”. That is the advantages of this method.

This method is the recommended one.

### 2.7.2 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.

---

---

**Note**

This feature is deprecated now und will be removed within the next releases.

---

### 2.7.3 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.

### 2.7.4 Automatically created wxApp instance

wxGlade can create an additional code to start an instance of projects “Top window”.

There are two types of application start code:

- simplified application start code
- detailed application start code

The application start code generation is controlled by three properties:

1. Name
2. Class
3. Top window

Those properties are explained in Section 3.4.1, “[Application Properties](#)”. Different combinations of those attributes generated different application start code. The table below shows the type of application start code resulting from different combinations of the three properties. The “Enable gettext support” property just triggers i18n-enabled source code.

Name	Class	Top window	Type of application start code to generate
not selected	not selected	not selected	not generated
selected	not selected	not selected	not generated
not selected	selected	not selected	not generated
selected	selected	not selected	not generated
selected	not selected	selected	simplified start code
not selected	selected	selected	not generated
selected	selected	selected	detailed start code

Table 2.1: Interaction between properties to generate different types of start code

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

---

```

1 package MyApp;
2 use base qw(Wx::App);
3 use strict;
4 sub OnInit {
5     my( $self ) = shift;
6     Wx::InitAllImageHandlers();
7     my $frame_1 = MyFrame->new();
8     $self->SetTopWindow($frame_1);
9     $frame_1->Show(1);
10    return 1;
11 }

```

---

```

12 # end of class MyApp
13 package main;
14 unless (caller) {
15     my $local = Wx::Locale->new("English", "en", "en"); # replace with ??
16     $local->AddCatalog("app"); # replace with the appropriate catalog name
17     my $app = MyApp->new();
18     $app->MainLoop();
19 }

```

---

### Example 2.3 Simplified application start code in Perl

```

1 package main;
2 unless (caller) {
3     my $local = Wx::Locale->new("English", "en", "en"); # replace with ??
4     $local->AddCatalog("PlOgg1_app"); # replace with the appropriate catalog name
5     local *Wx::App::OnInit = sub{1};
6     my $PlOgg1_app = Wx::App->new();
7     Wx::InitAllImageHandlers();
8     my $Mp3_To_Ogg = PlOgg1_MyDialog->new();
9     $PlOgg1_app->SetTopWindow($Mp3_To_Ogg);
10    $Mp3_To_Ogg->Show(1);
11    $PlOgg1_app->MainLoop();
12 }

```

## 2.7.5 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

```

## 2.8 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

---

```
1  #!/usr/bin/env python2
2
3  import wx
4  from wx import xrc
5
6  GUI_FILENAME = "test.xrc"
7  GUI_MAINFRAME_NAME = "Main"
8
9  class MyApp(wx.App):
10     def OnInit(self):
11         self.res = xrc.XmlResource(GUI_FILENAME)
12         self.frame = self.res.LoadFrame(None, GUI_MAINFRAME_NAME)
13         self.frame.Show()
14         return True
15
16 if __name__ == '__main__':
17     app = MyApp()
18     app.MainLoop()
```

---

## Chapter 3

# wxGlade User Interface

### 3.1 Main Palette

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

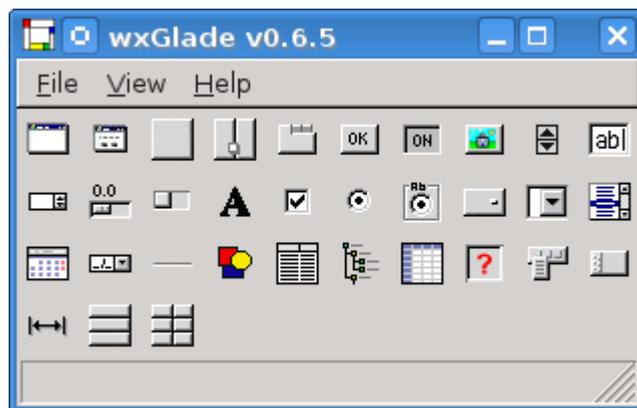


Figure 3.1: The Main Palette

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.

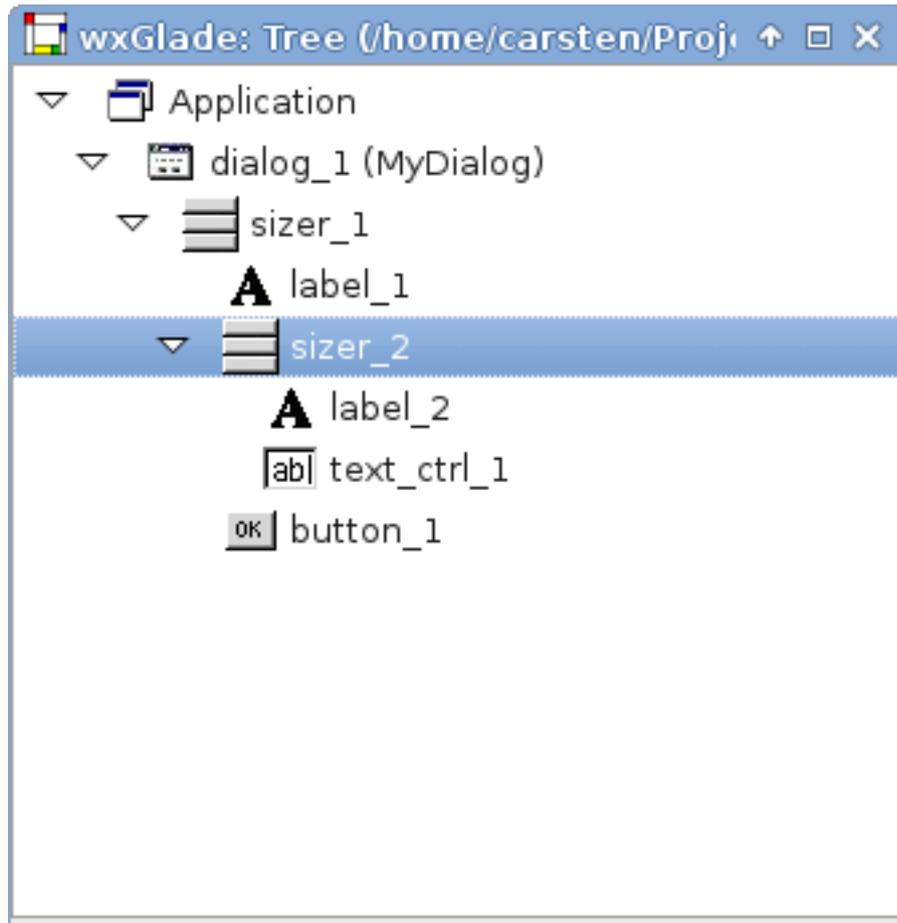


Figure 3.2: The Tree Window

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

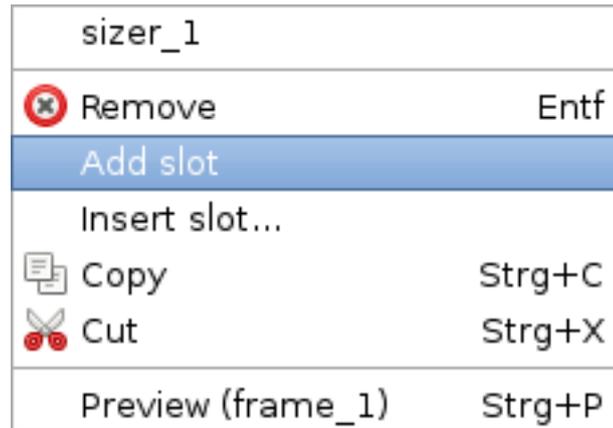


Figure 3.4: The menu for a sizer

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.

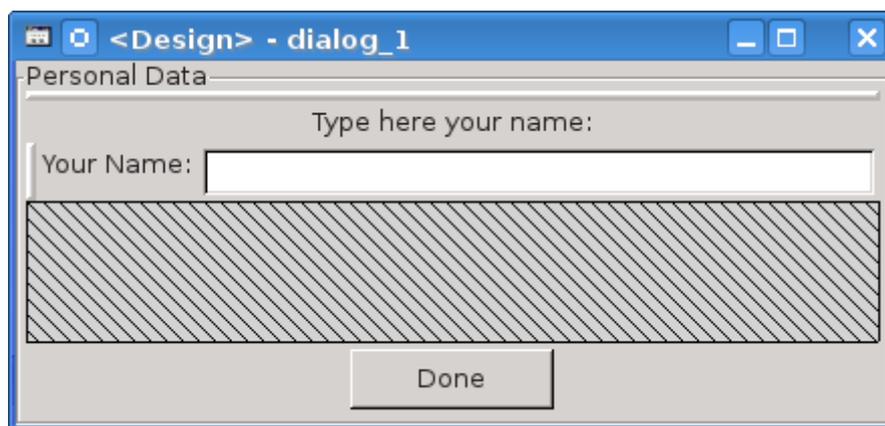


Figure 3.5: The Design Window

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, “The menu for a widget” or in Figure 3.4, “The menu for a sizer”.

---

## 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 Application Properties

The page “Application” contains the general settings of the active wxGlade project.

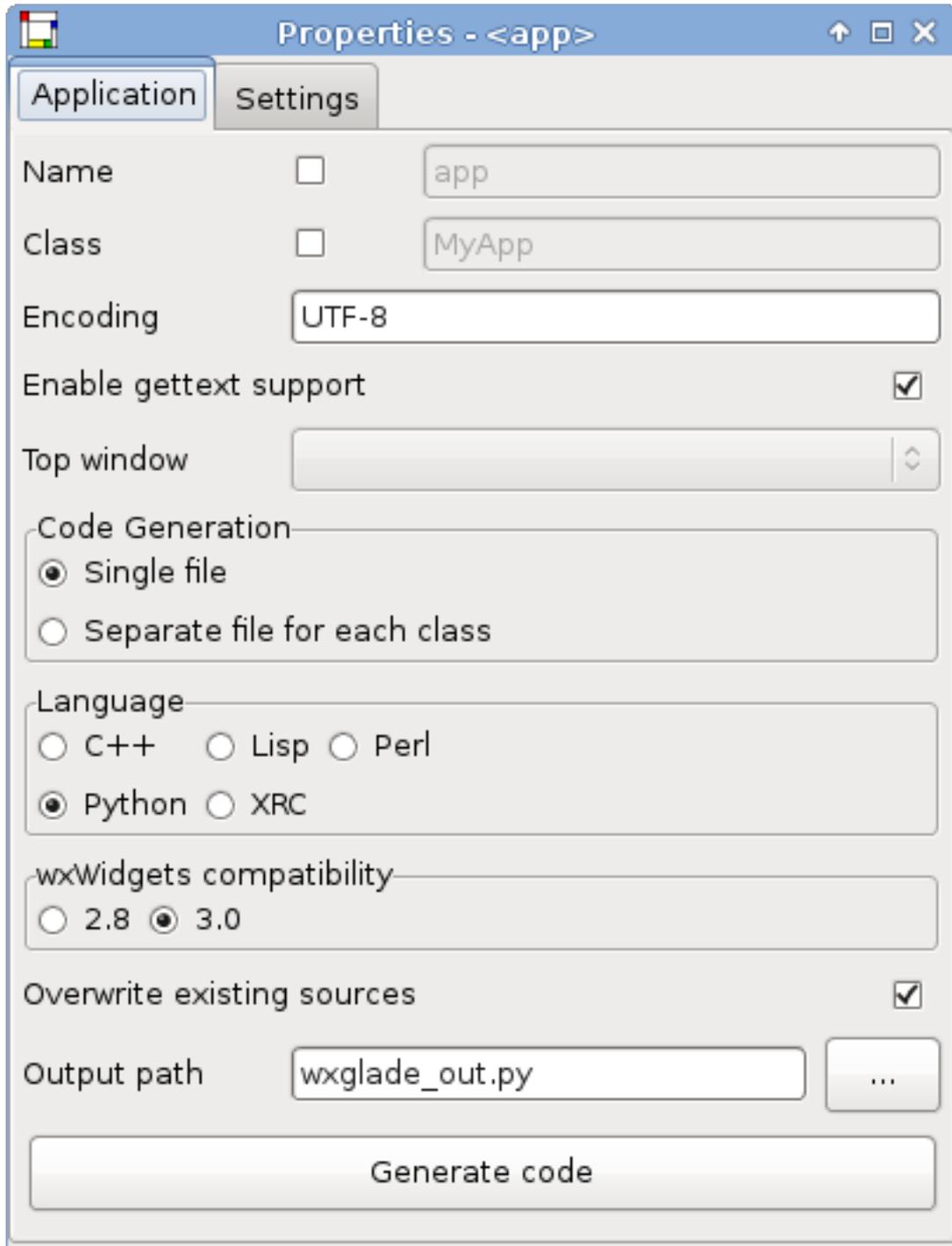


Figure 3.6: Project Properties - Application settings

#### “Name”

Name of the instance created from “Class”

Section 2.7.4, “Automatically created wxApp instance” provides more information

#### “Class”

Name of the automatically generated class derived from wxApp

Section 2.7.4, “Automatically created wxApp instance” 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

Section 2.7.4, “[Automatically created wxApp instance](#)” provides more information

**“Top window”**

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

Section 2.7.4, “[Automatically created wxApp instance](#)” provides more information

**“Code Generation”**

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

Section 2.7, “[Using the source code](#)” 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 2.7, “[Using the source code](#)” provides more information

---

**Note**

This feature is deprecated now und will be removed within the next releases.

---

**“Output path”**

Output file or directory

Section 2.7.3, “[Output path and filenames](#)” provides more information

**“Generate code”**

Start generating source files

The page “Settings” contains the language specific settings of the active wxGlade project.

---



Figure 3.7: Project Properties - Language settings

**“Indentation mode”**

Use spaces or tabs for indentation within the generated source files.

**“Indentation amount”**

Number of spaces or tabs used for one indentation level.

**“Source ext”**

Extension of the source file.

The extension doesn't have a leading dot.

### “Header ext”

Extension of the header file.

The extension doesn't has a leading dot.

## 3.4.2 Common Properties

The first tab contains the common properties that apply to all widgets. As shown in Figure 3.8, “Common Properties” the common properties are related to name, class, size, colors, fonts and tooltip.

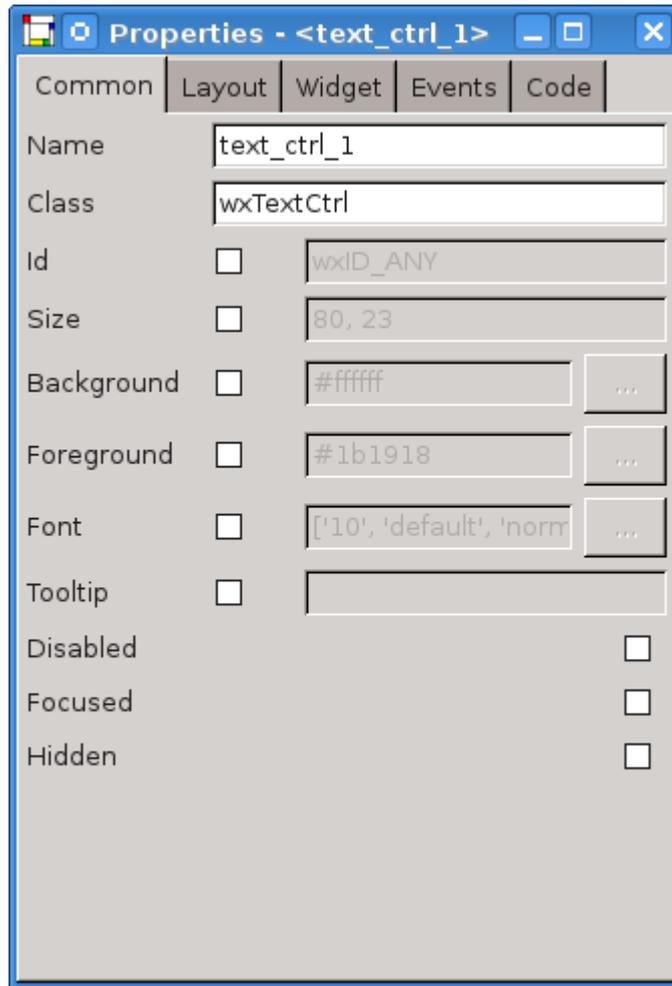


Figure 3.8: Common Properties

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 3.9, “Changing Common Properties”).

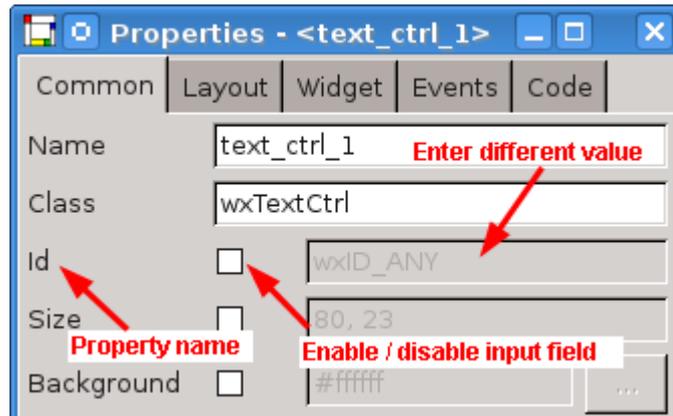


Figure 3.9: Changing Common Properties

**“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.

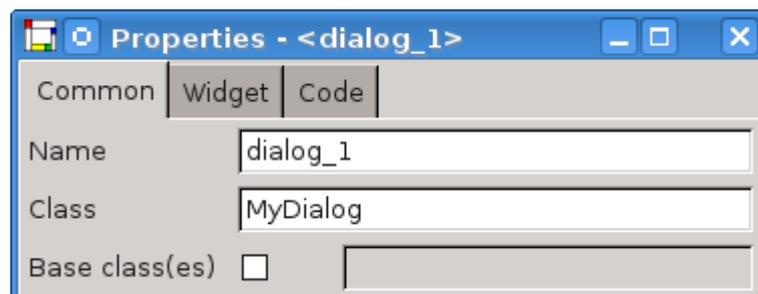


Figure 3.10: Common Properties of a subclassed widget (default behaviour)

**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"] = wxDEFAULT_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 3.11: 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()”

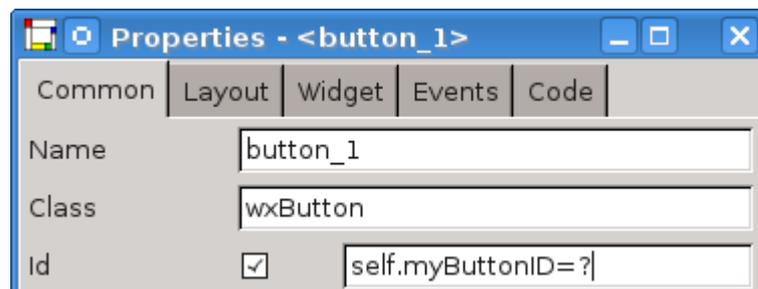


Figure 3.12: Common Properties with a variable assignment

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

```

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.myButtonID = wx.NewId()
7         self.button_1 = wx.Button(self, self.myButtonID, "button_1")
8         self.__set_properties()
9         self.__do_layout()
10        # 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.

### 3.4.3 Layout Properties

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

---

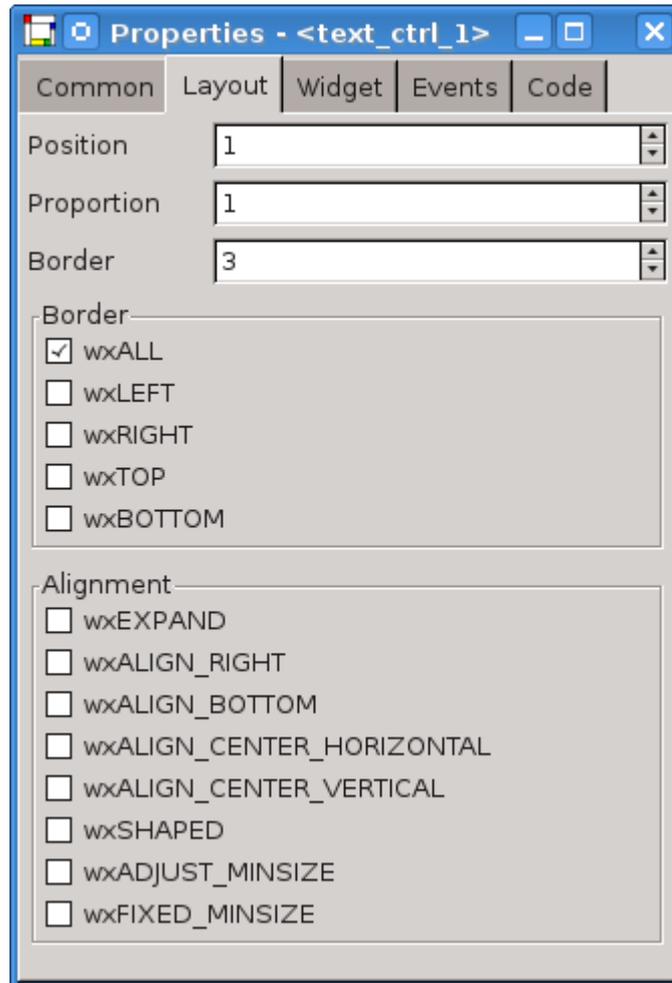


Figure 3.13: Layout Properties

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 1.6.1, “Preferences Dialog”). If you let a widget have a default border, the `wxALL` option is also checked.

### 3.4.4 Widget Properties

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

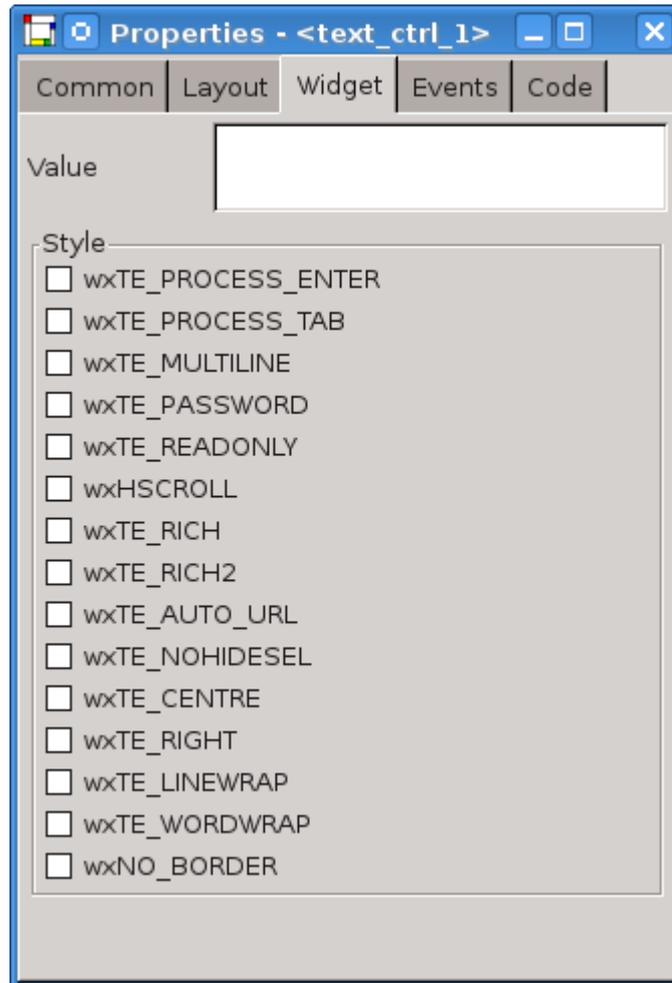


Figure 3.14: Widget Properties

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.

### 3.4.5 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.

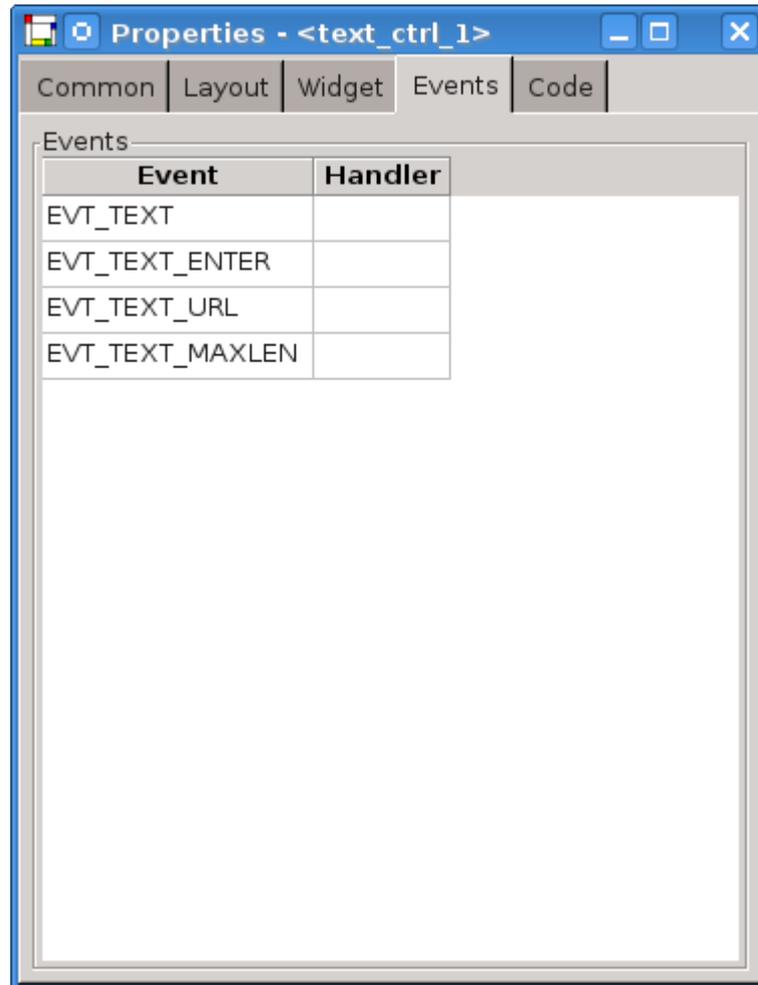


Figure 3.15: Events Properties

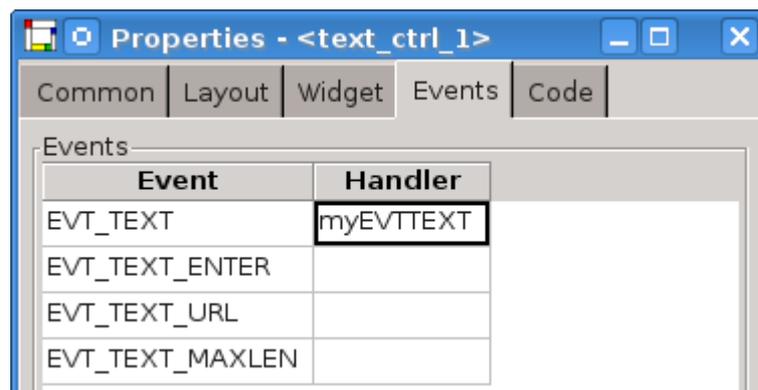


Figure 3.16: Events Properties with entered event handler name

**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()

```

### 3.4.6 Code Properties

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

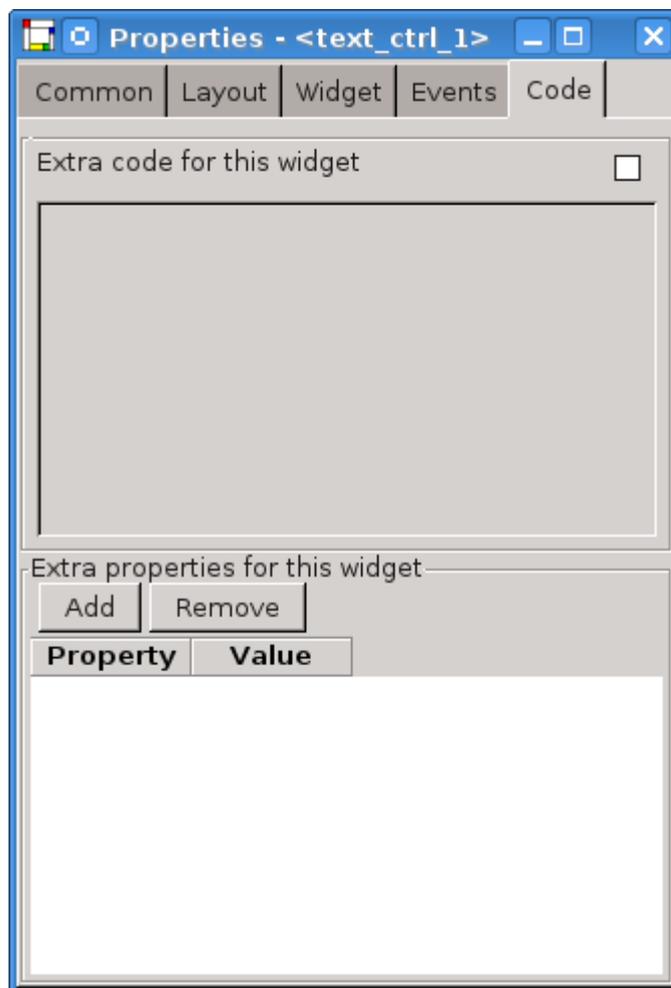


Figure 3.17: Properties for extra code and extra properties

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.

---

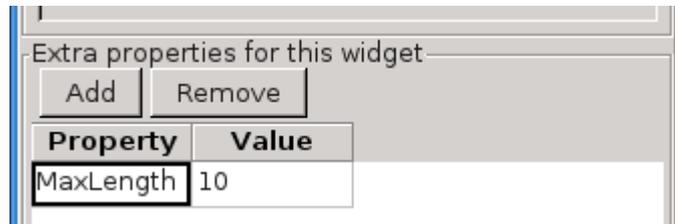


Figure 3.18: Set extra property

---

#### 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 The wxGlade Menu

wxGlade has only a few very small menus.

### 3.5.1 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.

### 3.5.2 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.

---

### 3.5.3 The HELP menu

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

## 3.6 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 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.

### 4.3 Widget List

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

#### 4.3.1 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.

#### 4.3.2 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.

---

### 4.3.3 Panel

This allows you to add a panel to a sizer.

In the properties window you can choose the styles.

### 4.3.4 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.

### 4.3.5 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.

### 4.3.6 Buttons

#### 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 4.3.6, "[Bitmap Button](#)").

#### 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 4.2, "[Specifying the path of bitmaps](#)" for bitmap path specifications.

#### Radio Button

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

#### Toggle Button

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

### 4.3.7 Gauge

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

### 4.3.8 Hyperlink Control

This produces a `wxHyperlinkCtrl`. In the property window you can enter the label, the URL and also set the style.

---

### 4.3.9 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.

### 4.3.10 Spin Control

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

### 4.3.11 Slider

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

### 4.3.12 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.

### 4.3.13 Text Control

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

### 4.3.14 Check Box

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

### 4.3.15 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.

### 4.3.16 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.

### 4.3.17 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.

### 4.3.18 Static Line

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.

---

### 4.3.19 Static Bitmap

This produces a `wxStaticBitmap`. You will be prompted for the bitmap path. Refer to Section 4.2, “[Specifying the path of bitmaps](#)” 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.

### 4.3.20 List Control

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

### 4.3.21 Tree Control

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

### 4.3.22 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.

### 4.3.23 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.

### 4.3.24 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, Statusbar and Toolbar

### 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.

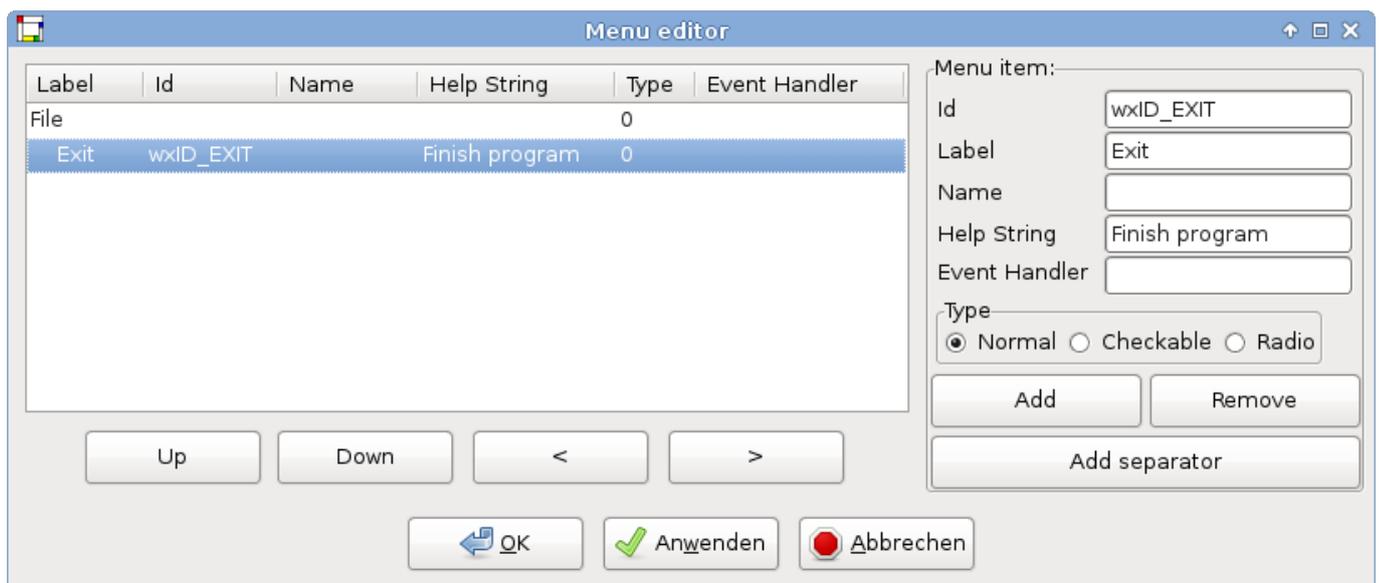


Figure 5.1: Menu editor

## 5.3 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.

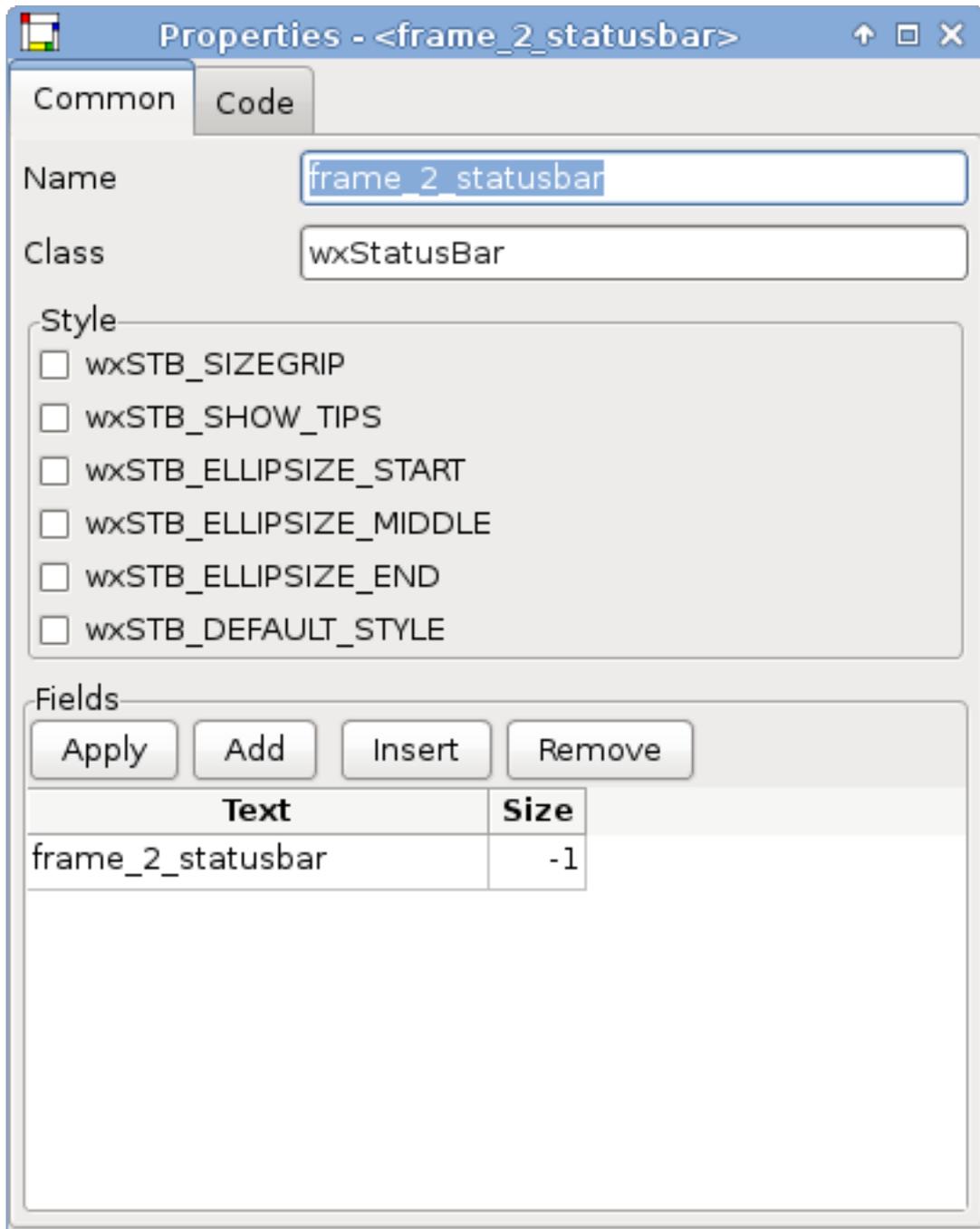


Figure 5.2: Statusbar properties

## 5.4 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 4.2, “[Specifying the path of bitmaps](#)” for bitmap path specifications.

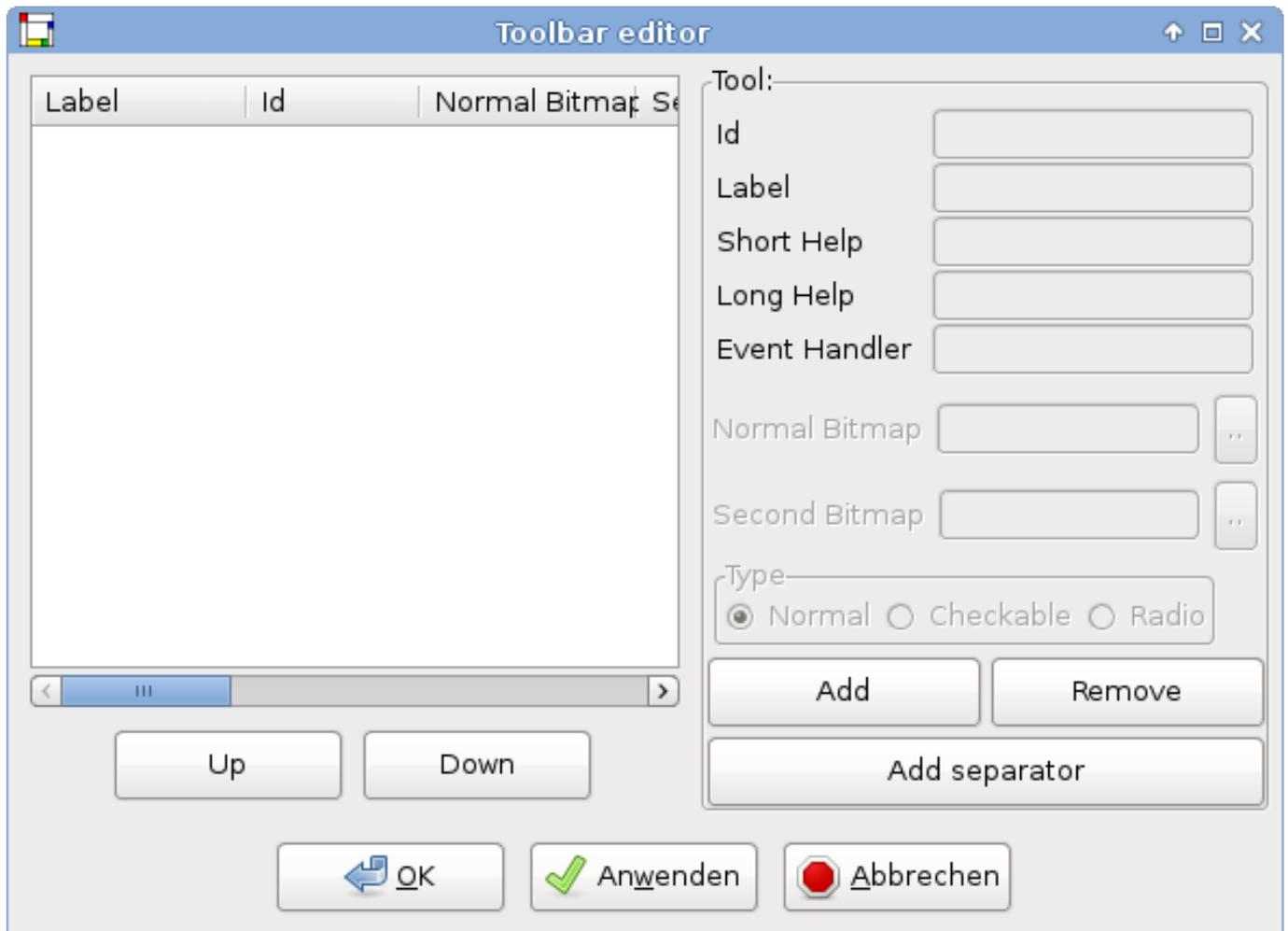


Figure 5.3: Toolbar editor

## Chapter 6

# wxGlade technical notes



### Caution

Last update: 2003-06-26, but only Section 6.9, “For contributors”.

The rest has not been updated since 2002-07-22, and it's likely be very outdated in some parts.

---

This is an informal overview of wxGlade internals, made through a sample session of use. Each action of the hypothetical user will be described from the point of view of the application, to (hopefully) understand what's happening behind the scenes.

These notes are *absolutely* incomplete and in some cases they might be outdated or not completely correct: the best reference is always the source code.

## 6.1 Startup

The program starts from the function “main” in the module “main”: this creates an instance of wxGlade (a subclass of wxApp), which in turn creates a wxGladeFrame: this is the main window of the app, i.e. the one with the palette of buttons. The initialization of wxGladeFrame consists of three steps:

- Creation of the three frames of the app, the palette itself, the tree and the property window
- Loading of the code generator modules. The “codegen/” subdir is scanned to find the available code generators: when a python module is found, the app tries to load it and to access its 'writer' attribute: if this is successfully accomplished, such 'writer' object is considered a valid code generator, and is inserted into the 'common.code\_writers' dict (the key used is the 'language' attribute of the writer itself)
- Loading of the widget and sizer modules. To load the widgets, the file “widgets/widgets.txt” is read, and the app tries to import every widget module listed on such file. For every module successfully imported, the “initialize” function is then called: this function sets up the builder and code generator functions for a particular widget (explained later), and returns a wxBitmapButton instance to be added to the main palette. The loading of the sizers is more or less the same, except that all the sizers are in the same module, “edit\_sizers”, and the initialization function (called “init\_gui”) returns a list of wxBitmapButton objects

## 6.2 Adding a toplevel widget

When the user clicks on a button of a toplevel widget (a Frame or a Dialog), the method “add\_toplevel\_object” of wxGladeFrame is called: this is responsible for the addition of the widget to the app. This happens in this way:

---

- the name of the class of the widget to add is obtained: this is done with the use of the “`common.refs`” dictionary, which maps the ids of the buttons of the palette to the class names of the widgets.
- with the name just obtained, the appropriate factory function for the widget to add is got from the “`common.widgets`” dictionary. This function must accept three parameters: a reference to the parent widget (`None` in this case), a reference to the sizer to which the widget will be added (again `None` for toplevel windows) and the zero-based position inside the sizer (once again, this is unused for toplevel windows)
- the call of the factory function actually builds the widgets and inserts it in the “`common.app_tree`” tree with a call to its method “`insert`”. The “`__init__`” method of the widget also builds all the Properties of the object and stores them in the `'self.properties'` dict

### 6.3 Adding a toplevel sizer

This is similar to the addition of a toplevel widget, but the action is performed in two steps:

- when the user clicks on the button in the palette, the method “`add_object`” of `wxGladeFrame` is called: this sets the global variables “`common.adding_widget`” and “`common.adding_sizer`” to `True`, and stores the class name of the sizer to add in the global “`common.widget_to_add`” (the name is obtained from the “`common.refs`” dictionary as described above)
- when the user left-clicks the mouse inside the previously added toplevel widget, its “`drop_sizer`” method is called, which is responsible of the addition of the sizer: it calls the factory function for the sizer (passing `self` as the first argument), which will build the object and add it to the tree

### 6.4 Adding a normal widget/sizer

This step is more or less the same as step 3:

- “`wxGladeFrame.add_object`” is called in response to a button click
- when the user “drops” the widget inside a slot in a sizer, the method “`drop_widget`” of `edit_sizers.SizerSlot` is called, which in turn calls the appropriate factory function with arguments “`self.parent`”, “`self.sizer`” and “`self.pos`” (i.e. the parent, sizer and position inside the sizer of the slot that will be replaced). Factory functions of non-toplevel objects call, apart from “`common.app_tree.insert`” to insert the object in the tree, the method “`add_item`” of “`edit_sizers.SizerBase`”, to add the object to the sizer and to remove the slot. For managed widgets/sizers, the “`__init__`” method also builds the Properties which control the layout of the object inside a sizer, and stores them in the “`self.sizer_properties`” dictionary.

### 6.5 Changing the value of a Property

When the user selects a widget the property window changes to display the properties of the selected object: this is done by the functions “`show_properties`” of `edit_windows.EditBase` and `edit_sizers.SizerBase`, which are called inside two event handlers for focus and tree selection events.

When the value of a Property is changed, its setter function is called to update the aspect/layout of the widget the Property belongs to: such function is obtained from a call to the widget’s “`__getitem__`” method, which must return a 2-tuple (getter, setter) for the Property

## 6.6 Saving the app

This operation is performed by the “`common.app_tree`” Tree: for every Node of the tree, an ‘object’ xml element is generated, with the following attributes: name, class, base (class). Each object contains an element for each Property (generated by the “`write`” method of Property) and then an ‘object’ element for all its sub-widgets and/or sizers. Properties in the “`sizer_properties`” dictionary are treated in a different way, as well as the children of a sizer, which are sub-elements of “`sizeritem`” objects: see the source code for details.

## 6.7 Loading an app from a XML file

This is done by “`xml_parse.XmlWidgetBuilder`”, a subclass of `xml.sax.handler.ContentHandler`.

Basically, the steps involved are the following:

- when the start of an ‘object’ element is reached, a `XMLWidgetObject` instance is created and pushed onto a stack of the objects created: such object in turn calls the appropriate “xml builder” function (got from the “`common.widgets_from_xml`” dictionary) that creates the widget: this function is similar to the factory function used to build the widget during an interactive session, see the code for details and differences
- when the end of an ‘object’ element is reached, the object at the top of the stack is removed, and its widget (see the source of `XmlWidgetObject`) is laid out
- when the end of a Property element is reached, the appropriate setter function of the owner of the Property is called. This is the default behaviour, suitable for simple properties. For more complex properties, whose xml representation consists of more sub-elements, each widget can define a particular handler: see for example `FontHandler` in `edit_windows.WindowBase`

## 6.8 Generating the source code

This section is the result of a cut & paste of the comment at the beginning of “`codegen/py_codegen.py`”. It is *\*VERY\** incomplete. The `ContentHandler` subclass which drives the code generation is `xml_parse.CodeWriter`.

How the code is generated: every time the end of an object is reached during the parsing of the xml tree, either the function “`add_object`” or the function “`add_class`” is called: the latter when the object is a toplevel one, the former when it is not. In the last case, “`add_object`” calls the appropriate “writer” function for the specific object, found in the “`obj_builders`” dictionary. Such function accepts one argument, the `CodeObject` representing the object for which the code has to be written, and returns 3 lists of strings, representing the lines to add to the “`__init__`”, “`__set_properties`” and “`__do_layout`” methods of the parent object.

---

**Note**

The lines in the “`__init__`” list will be added in reverse order.

---

## 6.9 For contributors

You are, of course, free to make any changes/additions you want to wxGlade, in whatever way you like. If you decide to contribute them back, however, here are some simple (stylistic) rules to follow: note that these are only general indications, if you think they don’t fit somewhere, feel free to ignore them.

- class names are usually CamelCase - variables, functions and method names are lower\_case\_with\_underscores
  - “constants” are UPPER\_CASE
  - source lines are at most 79 characters long
-

- class bodies are usually ended by a “#end of class ClassName” comment
  - source files use Unix EOL conventions (LF) if possible. In any case, please don't mix Unix and Windows EOLs
  - put your copyright info whenever appropriate
  - that's all folks!!
-

## Chapter 7

# Installing and Designing own Widget Plugins

wxGlade supports a simple plugin system for widgets to load all widgets at the application startup dynamically. The plugin system loads all

built-in widgets like “Static Text” widget or the “Gauge” widget. It also loads widgets installed by users.

### 7.1 Widgets packages

The wxGlade plugin system supports two different types of widget packages:

1. “directory package” - a single directory with all necessary files inside
2. “ZIP package” - a zipped version of a "directory" package

---

#### Example 7.1 Directory package

```
static_text      <- Directory named after the widget name
|-- __init__.py  <- Mostly an empty file or a file with just a comment
|-- codegen.py   <- Python and C++ code generators
|-- wconfig.py   <- Widget configuration
|-- lisp_codegen.py <- Lisp code generator
|-- perl_codegen.py <- Perl code generator
`-- static_text.py <- wxGlade GUI code
```

---



---

#### Example 7.2 ZIP package

```
# unzip -l static_text.zip
Archive:  static_text.zip
  Length      Date    Time    Name
-----
         0  2013-12-09 10:02  static_text/
        329  2013-12-09 10:02  static_text/__init__.py
       3352  2013-12-09 10:02  static_text/codegen.py
        320  2013-12-09 10:02  static_text/wconfig.py
       1640  2013-12-09 10:02  static_text/lisp_codegen.py
       1841  2013-12-09 10:02  static_text/perl_codegen.py
       5917  2013-12-09 10:02  static_text/static_text.py
-----
      13079
                   6 files
```

---

### 7.1.1 Create ZIP package

Creating a ZIP package is quite simple. Just create a ZIP package from widgets directory with all Python and additional files. Don't include Python bytecode files because they are not platform-independent.

```
# tree static_text/
static_text/
|-- __init__.py
|-- codegen.py
|-- wconfig.py
|-- lisp_codegen.py
|-- perl_codegen.py
`-- static_text.py

# zip -r static_text.zip static_text
adding: static_text/ (stored 0%)
adding: static_text/__init__.py (deflated 36%)
adding: static_text/codegen.py (deflated 67%)
adding: static_text/wconfig.py (deflated 64%)
adding: static_text/lisp_codegen.py (deflated 54%)
adding: static_text/perl_codegen.py (deflated 56%)
adding: static_text/static_text.py (deflated 69%)
```

Check the integrity of the created ZIP archive:

```
# zip -T static_text.zip
test of static_text.zip OK
```

## 7.2 Installing Widget Plugins locally

The installation of local plugins is a two-step process:

1. Place the widget package in the Local widget path (see Section 1.6.1, “Preferences Dialog”). Create this directory if it doesn't exist.
2. Add widget name to the text file named `widgets.txt`. This file is also located in the directory specified in Local widget path. Just create a simple text file, if the file doesn't exist.

The new widget will be available after myrun has been restarted.

## 7.3 Designing own Widget Plugins

---

### Note

This section is under construction! Please use this information carefully.

---

1. Create a new directory named like the widget and change in this directory
2. Place an empty file `__init__.py` in that directory
3. Create a Python file `codegen.py` with initial content like

```
1  """
2  Code generator functions for myCtrl objects
3
4  @copyright: <Add year and your name>
```

---

```
5 @license: <Choice a license>
6 """
7
8 import common
9
10
11 class PythonMyCtrlGenerator(wcodegen.PythonWidgetCodeWriter):
12
13     tmpl = '%(name)s = %(klass)s(%(parent)s, %(id)s, %(label)s%(style)s)\n'
14
15 # end of class PythonMyCtrlGenerator
16
17
18 def initialize():
19     common.class_names['EditmyCtrl'] = 'myCtrl'
20
21     pygen = common.code_writers.get("python")
22     if pygen:
23         pygen.add_widget_handler('myCtrl', PythonMyCtrlGenerator())
```

4. Create a Python file named like the widget directory e.g. myctrl.py
5. Create remaining code generators
6. Example of the created structure

```
myctrl
|-- __init__.py
|-- codegen.py
`-- myctrl.py
```

### 7.3.1 Widget Initialisation

XXX

1. Load generic and language independent widget configuration from wconfig.py (`common.load_config()`)
2. Load and initialise language code writers (`common.load_code_writers()`)
3. Load and initialise widgets (`common.load_widgets()`)
4. Load and initialise sizers (`common.load_sizers()`)

XXX

---

## Appendix A

# Abbreviations

The following abbreviations are used in this manual:

**Escape sequence** Escape sequences are used to define certain special characters within string literals. Escape sequences starts mostly with a backslash (“\”).

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

**GUI** Graphical User Interface

**i18n** Numeronyms for internationalisation support.

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

**OS** Operating system

**OS X** is a graphical UNIX operating system developed by Apple Inc. OS X is certified by The Open Group.

**SAE** Standalone Edition

**Unix** is a multitasking, multi-user operation system. Today the trademark “UNIX” is owned by The Open Group. Operating systems complaint with “Single UNIX Specification” and certified by The Open Group are called “UNIX”.

Unix-like operating systems behaves similar to “UNIX” operating systems. They are not certified by The Open Group.

This document use the term “Unix” for certified “UNIX” operating systems as well as “Unix-like” operating systems. Linux is an Unix-like operating system.

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

**wx** abbreviation for wxWidgets

**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.

---

## Appendix B

# 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 C, [wxGlade License Agreement](#).

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

Microsoft and Windows are registered trademarks of Microsoft Corporation.

UNIX is a registered trademark of The Open Group.

All other trademarks are property of their respective owners.

---

## Appendix C

# wxGlade License Agreement

Copyright (c) 2002-2007 Alberto Griggio

Copyright (c) 2011-2014 Carsten Grohmann

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 D

# Licenses and Acknowledgements for Incorporated Software

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

### D.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.

---