wxGlade manual

Contents

List of Figures

List of Tables

List of Examples

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

Chapter 1

Getting Started

1.1 What is wxGlade?

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

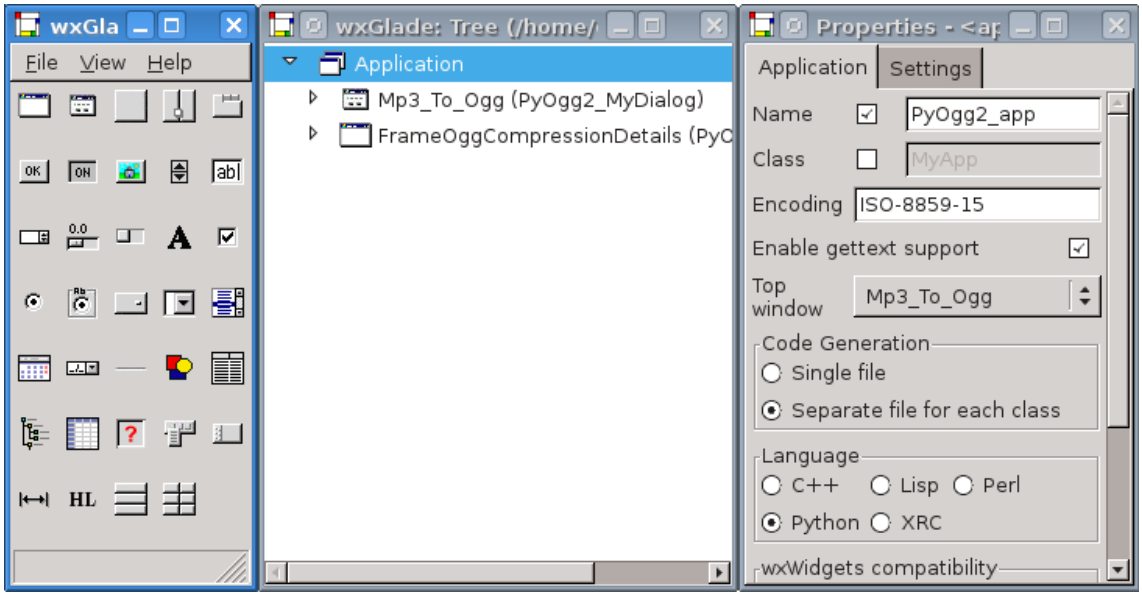


Figure 1.1: wxGlade windows

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).

1.1.1 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

1.1.2 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 Eric Python IDE, PyCharm, Code::Blocks or one of the many other IDEs 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.

Chapter 2

Installation

2.1 Requirements and Supported Platforms

wxGlade has been run on Microsoft Windows, Linux, OS X because it's written in Python using wxPython. wxGlade can be run on any platform that supports Python and wxPython.

The requirements to run wxGlade 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.

2.2 Installation

wxGlade is available in four different package types:

1. the sources packages (`.zip` and `.tar.gz`)
2. the full installer on Microsoft Windows (`wxGlade-<VERSION>-setup.exe`)
3. the installer of the Standalone Edition on Microsoft Windows (`wxGlade-SAE-<VERSION>-setup.exe`)
4. current development version

2.2.1 Requirements for Building Own Packages

You need listed additional packages for building wxGlade packages from the Mercurial repository:

- setuptools

2.2.2 Download

Official Release Packages

All stable version are available for downloading at https://sourceforge.net/projects/wxglade/.

Development Versions

wxGlade manages source code in a Mercurial repository hosted on Bitbucket.org. You can fetch the whole repository from https://bitbucket.org/wxglade/wxglade using anonymous Mercurial (**hg**) access.

Alternatively you can download different source tarballs from https://bitbucket.org/wxglade/wxglade/downloads.

2.2.3 Installing on Microsoft Windows

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

The Standalone Edition doesn't need additional packages. It's includes already the required parts of Python, wxPython and wxWidgets.

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

2.2.4 Installing on 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.

Install wxGlade from the source package like described in Section 2.2.5, "Installing from Source" if your distribution doesn't provide any package or the package is out-of-date.

On Unix systems wxGlade will be started using the shell script **wxglade**. The script search the system for Python 2 and finally executes the Python interpreter to start wxGlade.

You can set the environment variable `PYTHON_BIN`, if the shell script **wxglade** doesn't find Python or if you want to use a non-default interpreter.

2.2.5 Installing from Source

The installation from scratch requires Python, wxPython and wxWidgets. Install these three components first. You can use already packaged versions of those components for your operating system. Otherwise read the installation documentation of the missing components and follow the instructions.

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

You can install wxGlade from source in three different ways.

Single User Installation (Extract and Run)

> In short:
>
> Just download the archive of current version as a .zip file or a .tar.gz file, extract the archive into a empty directory and execute **wxglade** on Unix or **wxglade.pyw** on Windows to start wxGlade.

The "Single User Installation" is a simplified installation without using native package like `.msi` or `.exe` files on Microsoft Windows or `.rpm`, `.pkg` and `.deb` on Linux. It's to provide a simple way for running wxGlade without the need of installation privileges.

The dependencies listed in Section 2.1, "Requirements and Supported Platforms" have been satisfied.

The source packages and the development packages have the same structure. They have a top-level directory and all wxGlade files are under those top-level directory. Extract the package into the user's home directory. Take care that the directory structure of the archive will be preserved. Rename the top-level directory to wxGlade. That's shorter and simpler to use.

Change into the new created directory. Execute the **wxglade** file on Unix operating systems or **wxglade.pyw** on Microsoft Windows to start wxGlade. Feel free to create a shortcut on your desktop.

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

Multi User Installation - Variant A on Windows and Unix

The first variant of a multi user installation is very similar to Section 2.2.5, "Single User Installation (Extract and Run)" except the installation directory. And probably you need administrative permissions. You could extract the wxGlade source package e.g. into **C:\Program Files\wxGlade** on Microsoft Windows or into **/opt/wxGlade** on Unix. Execute the **wxglade** file on Unix operating systems or **wxglade.pyw** on Microsoft Windows to start wxGlade.

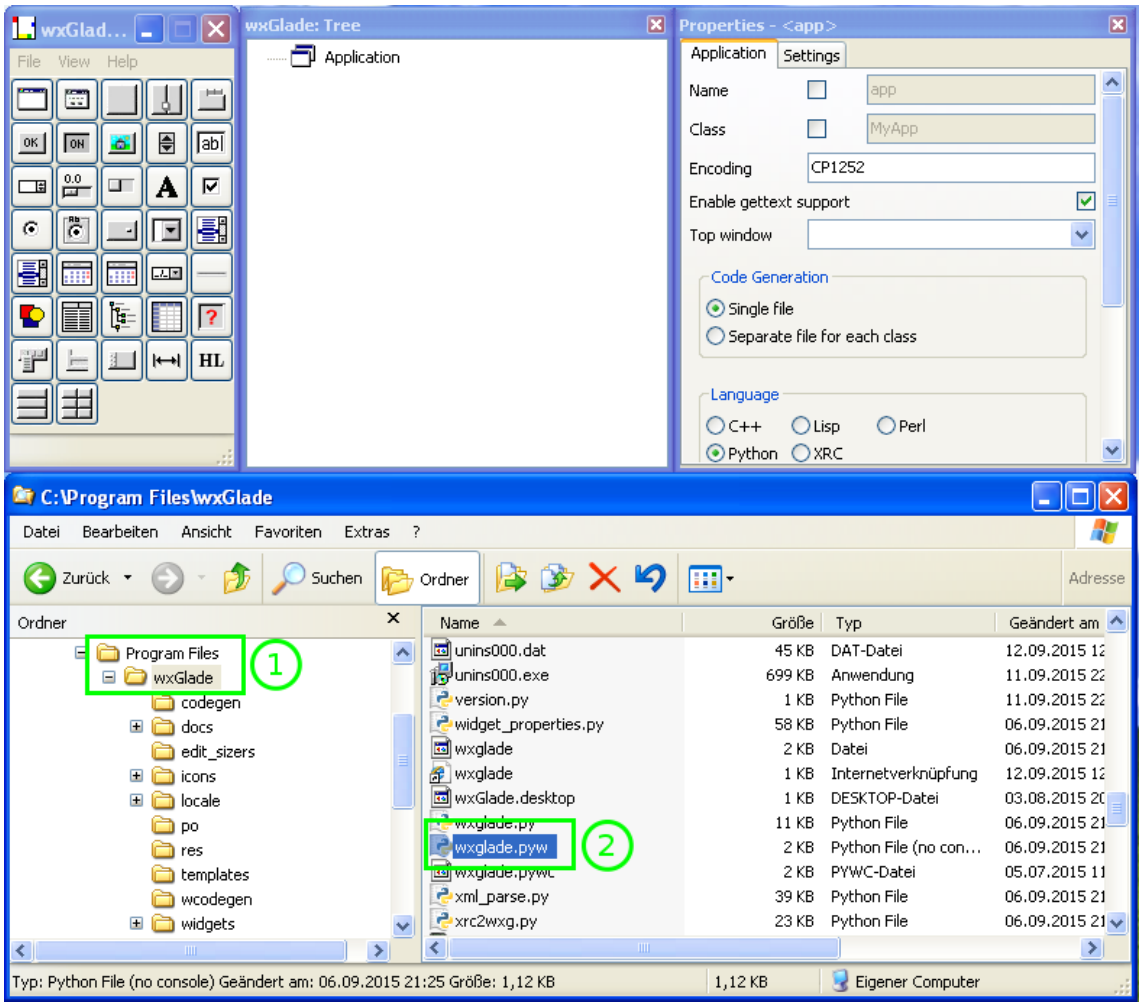


Figure 2.1: Starting wxGlade on Windows at C:\Program Files\wxGlade

Multi User Installation - Variant B on Unix

The second installation method is only supported Unix operating systems.

Extract the downloaded package into a temporary directory. Open a terminal window, change into the temporary directory and install wxGlade using execute the Python setup script **setup.py** as follow:

python2 setup.py install --root <DESTINATION DIRECTORY>

Note
Parameters shown in angle brackets (e.g., "*<parameter>*") are required.
Parameters shown in square brackets (e.g., "*[parameter]*") are optional. If not given, wxGlade will use suitable defaults.

Example 2.1 Installing wxGlade at /opt/wxglade

```
# python2 setup.py install --root /opt/wxGlade
running install
running build
running build_py
running build_scripts
running install_lib
creating /opt/wxGlade
creating /opt/wxGlade/usr
creating /opt/wxGlade/usr/lib
creating /opt/wxGlade/usr/lib/python2.7
[...]
writing manifest file 'wxGlade.egg-info/SOURCES.txt'
Copying wxGlade.egg-info to /opt/wxGlade/usr/lib/python2.7/site-packages/wxGlade-0.7.1rc1- ↩
    py2.7.egg-info
running install_scripts
creating /opt/wxGlade/usr/bin
copying build/scripts-2.7/wxglade -> /opt/wxGlade/usr/bin
changing mode of /opt/wxGlade/usr/bin/wxglade to 755
```

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

Execute the script **<install directory>/usr/bin/wxglade** to start wxGlade

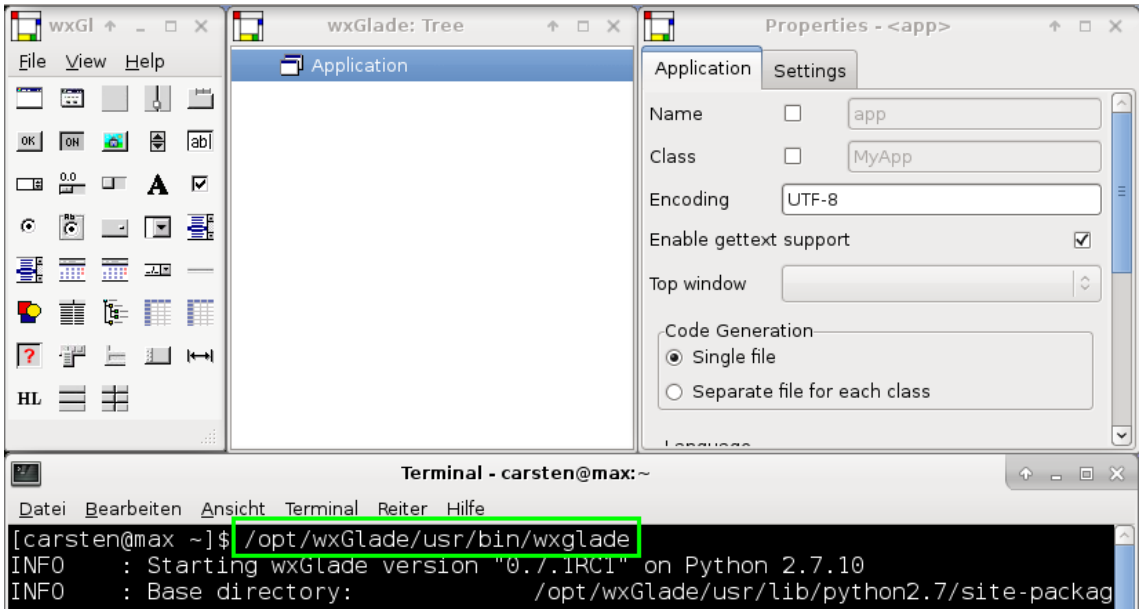


Figure 2.2: Starting wxGlade on Linux at /opt/wxglade/bin/wxglade

Chapter 3

Running wxGlade

Once the installation of wxGlade is finished successfully, you can start wxGlade either by clicking on the desktop icon, selecting the program from the “Start Menu” or on a command line as described in Section 3.3, “Command Line Invocation”.

3.1 Known Platforms

wxGlade works on most Unix platforms and Microsoft Windows.

3.2 Languages

wxGlade uses English only, currently.

3.3 Command Line Invocation

You can run wxGlade without parameters to start the GUI with an empty project as follows:

wxglade

Run wxGlade GUI on an existing project 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.

Complete command line description:

```
# 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 output.py -g python myapp.wxg

Report bugs to:    <wxglade-general@lists.sourceforge.net> or at
                   <https://sourceforge.net/projects/wxglade/>
wxGlade home page: <http://wxglade.sourceforge.net/>
```

Note
Use **wxglade.pyw** instead of **wxglade** on Microsoft Windows.
Parameters shown in angle brackets (e.g., "*<parameter>*") are required.
Parameters shown in angle brackets (e.g., "*<parameter>*") are required.

3.4 Configuring wxGlade

3.4.1 Preferences Dialog

You can access the Preferences Dialog with the menu View → Preferences. You can choose some decoration options, like whether to show icons in menus or not. 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.

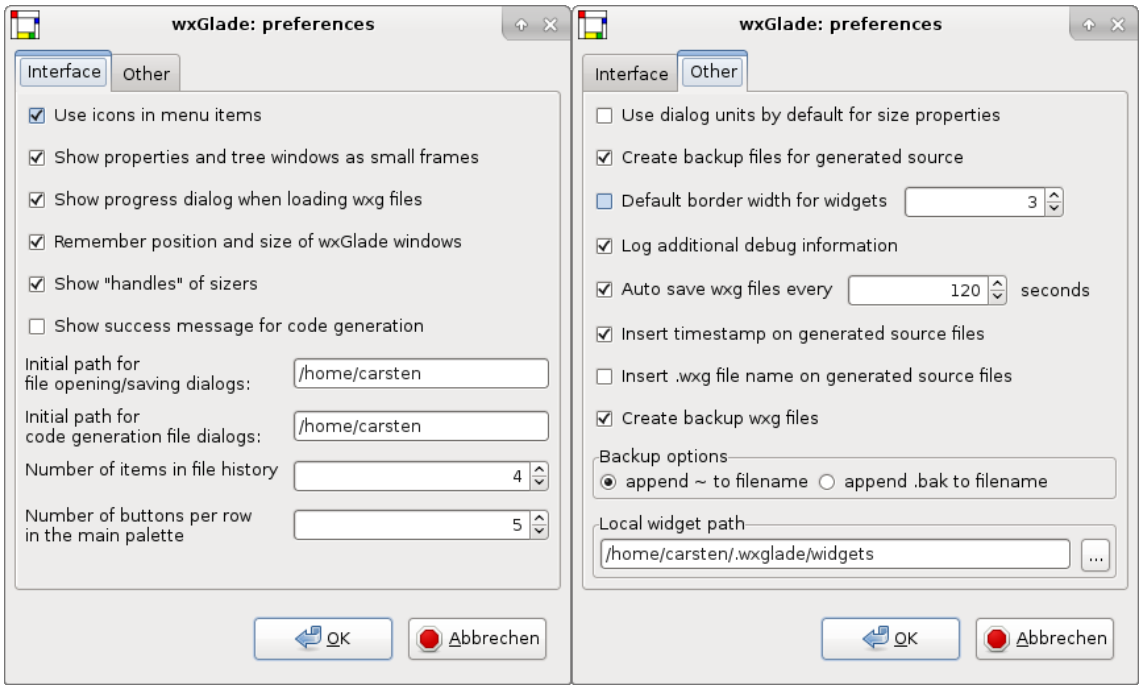


Figure 3.1: wxGlade preferences dialog

3.4.2 Environment Variables

wxGlade supports 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.

3.4.3 Configuration Files

wxGlade stores several internal information like configuration settings and log files in an own directory. The location of this directory varies between different operating systems. On Unix wxGlade uses `~/.wxglade`. On Microsoft Windows the information will be stored in `C:/Users/<username>/AppData/Roaming`.

Note
Some components of the path will be translated to the users’ language on Microsoft Windows automatically and transparently in the Windows Explorer and other graphical applications. Other applications show the original names. The translation behaviour depends on the API used by the application developer.

The configuration file is a simple text file and contains user settings. It’s named `.wxgladerc` on Unix or `wxglade.ini` on Microsoft Windows. Be careful with manual changes.

The configuration directory contains also a list of recently used files in `file_history.txt`.

wxGlade writes always a small error log file `wxglade.log`. It’s also stored inside this directory. The file size is limited to 100 KB and 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 an UTF-8 encoded text file.

3.5 Deprecated Features

This section lists deprecated features of wxGlade. These features will be removed from a future version of wxGlade.

3.5.1 “Overwrite existing sources”

wxGlade can change the own code inside existing source files to reflect changed designs. This feature have some limitations e.g. in case of name and dependencies changes. Thereby it’s recommended to overwrite existing sources always and extend derived classes with your functionality.

The pro and cons are discussed detailed in Section 4.2.4, “Shared Control”.

This feature isn’t removed currently to stay backward compatible.

Warning
Don’t use this feature for new projects.

3.6 Troubleshooting and Error Handling

3.6.1 Troubleshooting

This section should help you to solve some non-application issues.

wxGlade has started e.g. by clicking on the desktop icon, but no application window opens

Check the log file for errors (see Section 3.4.3, “Configuration Files”) and check that all dependencies (especially wxWidgets and wxPython) are installed. A list of dependencies is in Section 2.1, “Requirements and Supported Platforms”.

If you think this is an application issue, please file a bug report like described in Section 3.6.3, “How to Report a Bug”.

3.6.2 Error Messages

The following list of error messages should help you to find the reason and take corrective actions. Please note, that the list is still incomplete.

Output path "<path>" must be an existing directory when generating multiple files.

Details: You want to create multiple files form your design. wxGlade expects an output directory in such case, but the given output path isn’t a existing directory.

Corrective Action: Create the missing directory or select an existing output directory.

Output path "<path>" exists but the directory is not writable.

Corrective Action: Choose a writable output directory.

Output path "<path>" can not be a directory when generating a single file.

Details: You want to generate a single source code files for your design. wxGalde expects an output filename, but the given output path is a directory.

Corrective Action: Choose a file instead of a directory.

Output path "<path>" should be a directory when generating multiple files.

Details: You want to generate multiple source code files for your design. wxGlade expects an output directory in such case, but the given output path isn’t a directory.

Corrective Action: Choose an existing output directory.

Generated source code couldn't converted to encoding <encoding>. The source contains invalid characters "<characters>" fro

Details: The current design uses characters that can't converted to the encoding selected for the source code file.

Corrective Action: Choose a encoding that supports the listed characters.

Generating Lisp code for wxWidgets version <version> is not supported.

Details: There is no Lisp support for the selected version of wxWidgets

Corrective Action: Create the code for an older wxWidget version.

XRC code cannot be split into multiple files.

Details: You want to create multiple XRC files for your design. This option isn't supported by wxGlade.

Corrective Action: Select the "Single file" option to store the whole design in one XRC file.

Code generation from a template is not possible.

Details: You've selected a template as input file to generate source code from.

Corrective Action: Use a design file instead of a template.

Code writer for "<language>" is not available.

Details: The code generators for the given language hasn't been loaded.

Corrective Action: That's probably a bug in wxGlade. Please check the log file and file a bug report. See Section 3.6.3, "How to Report a Bug" for more details.

Conversion of the source file <filename> to Unicode failed.

Details: The file encoding set within the design doesn't match the encoding used within the source file. Thereby the source file can't be loaded.

Corrective Action: Check the encoding selected in your design and the encoding of your source file. Change the design encoding to match source files encoding.

Please install missing Python module "wxPython".

Details: The Python wrapper for wxWidgets couldn't be loaded.

Corrective Action: Check if

- you have wxPython installed
- the wxPython version matches to the Python installation
- Python and wxPython used the same memory address width of 32-bit or 64-bit

Please install missing Python module "wxversion".

Details: The Python module "wxversion" couldn't be loaded.

Corrective Action: "wxversion" is a Python module written in pure Python. Thereby you can't have trouble with 32bit and 64bit like with "wxPython". Just install the Python module "wxversion". Mostly the module is part of the wxPython installation.

Too many wxg files given.

Details: You want to start the code generation from command line, but you have chosen to many design files.

Corrective Action: Re-execute the command with the just design one file name as last argument.

No wxg file given.

Details: You want to start the code generation from command line, but you haven't specify the design file to process.

Corrective Action: Re-execute the command with the design file name as last argument.

3.6.3 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 absolute rules - it's more like a guideline. Additionally you may read How to Report Bugs Effectively.

- What did you? May you want to include a screen shot.
- 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. The pane "How to Report a Bug" contains the full path of the error log file. The file location is additionally described in Section 3.4.3, "Configuration Files".

Please create a new bug report in the wxGlade bug tracker on Sourceforge. You have to log-in on Sourceforge to file a bug report.

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.

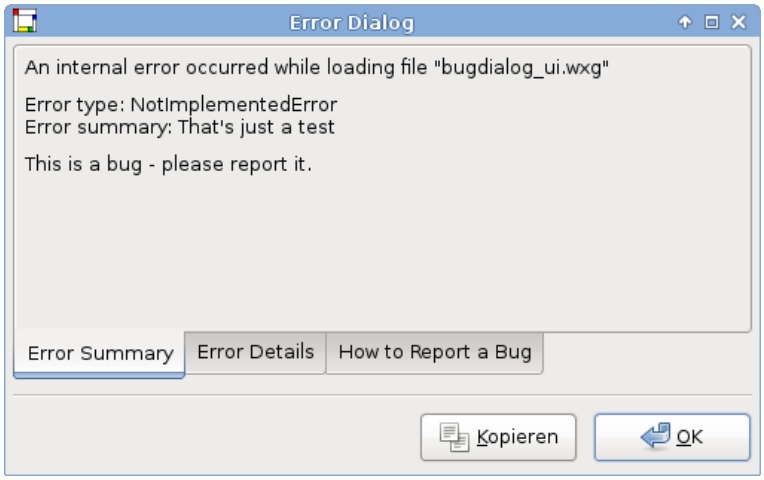


Figure 3.2: An error dialog example

Chapter 4

Working with 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 on 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.

4.1 First Steps

These sections provided you a short introduction into wxGlade. Advanced topics like escaping special characters or naming convention will be discussed in Section 4.2, "Next Steps".

The used abbreviations are described in Appendix A, Glossary of Terms, Abbreviations, and Acronyms.

4.1.1 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 Palette, 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" and 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 on the frame icon, the designer window appears. Notice that the sizer is displayed as a set of gray hatched boxes: they are the "slots" of the sizer where you will place the widgets.

You place a widget on a sizer by selecting it on the Main Palette, 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 and the “Widget” tab enables you to change the properties of 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.

4.2 Next Steps

This section covers advanced topics like the handling of automatically generated source files or escaping of special characters like newline characters or tab characters.

4.2.1 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 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 exact the same sequence and don’t convert it into control characters. Question marks especially double quotes will be escaped also.

4.2.2 Best Practice

These recommendations should help to improve the usability and maintainability of code generated by wxGlade. They combine the experience of many wxGlade users.

Overwrite Existing Sources

wxGlade is able to change the own code inside existing source files to reflect changed designs.

This feature has some limitations. They are detailed discusses in Section 4.2.4, “Shared Control”.

It’s common coding practice to separate and encapsulate individual responsibilities. Using a derived class to extend the code e.g. with GUI logic would follow this practice.

Warning
This feature is deprecated and will be removed from a future version of wxGlade. Thereby it’s advised not to use this feature for new projects. Details about all deprecated features are shown in Section 3.5, “Deprecated Features”.

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 4.1 Correct entered wx constant

Enter "**wxID_CANCEL**" even for wxPython instead of "**wx.ID_CANCEL**"

Use UTF-8 encoding

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

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.

Naming Conventions

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.

4.2.3 Language Specific Peculiarities

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 4.2.2, "Best Practice" also.

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

XRC

The XRC code writer doesn't supports all bitmap path tags described in Section 6.2, "Specifying the Path of Bitmaps".

Just the "`art:`" statement is supported. The remaining bitmap tags will irgnored.

4.2.4 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 5.6, “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.

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 4.2.4, “Shared Control”. That is the advantages of this method.

This method is the recommended one.

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.

The source code modifications by wxGlade may incomplete after:

- renaming classes and attributes
- changes in dependencies are not updated in all use cases
- changing the base classes or replace a single base class by multipe base classes or vis-a-vis
- nested classed and functions
- if the percent sign (“%”) have been added manually
- removing event handlers

Additionally there are some internal flaws like the inconsistent design of the begin and end markers, just compare the event handler code generated for Perl and Python. Improving the handling internally would break the backward compatibility.

Warning
This feature is deprecated and will be removed from a future version of wxGlade. Thereby it’s advised not to use this feature for new projects. Details about all deprecated features are shown in Section 3.5, “Deprecated Features”.

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”).

Automatically Created `wxApp` Instances

wxGlade can create 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 5.4.1, "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 4.1: 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 4.2 Detailed application start code in Perl

```
1  package MyApp;
2
3  use base qw(Wx::App);
4  use strict;
5
6  sub OnInit {
7      my( $self ) = shift;
8
9      Wx::InitAllImageHandlers();
10
11     my $frame_1 = MyFrame->new();
12
13     $self->SetTopWindow($frame_1);
14     $frame_1->Show(1);
15
16     return 1;
17 }
18 # end of class MyApp
19
20 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 4.3 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();
}
```

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 4.4 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 4.5 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
```

4.2.5 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 4.6 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 4.7 wxPython code to load and show a XRC resource

```
#!/usr/bin/env python2

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()
```

Chapter 5

wxGlade User Interface

5.1 Main Palette

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

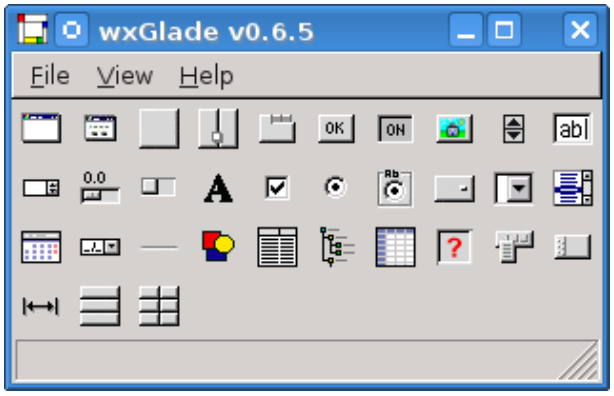


Figure 5.1: The Main Palette

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.

5.2 Tree Window

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

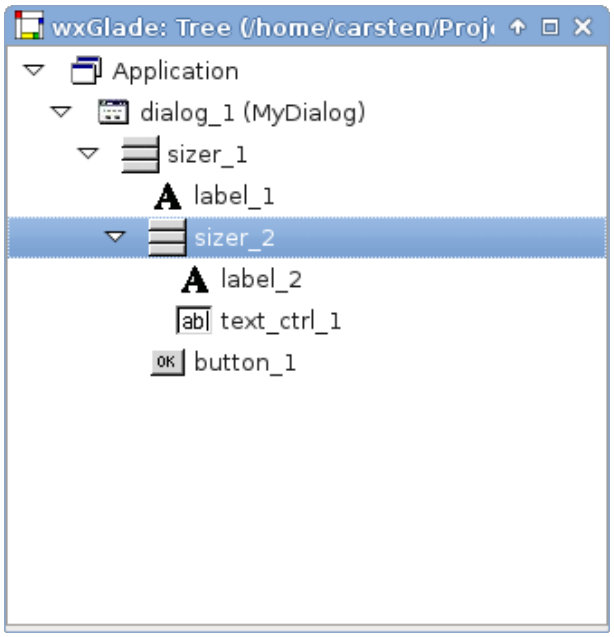


Figure 5.2: The Tree Window

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 open the Design Window with double-clicking to the top-level widget first. In the next step 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.

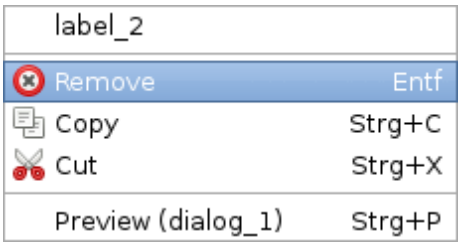


Figure 5.3: The menu for a widget

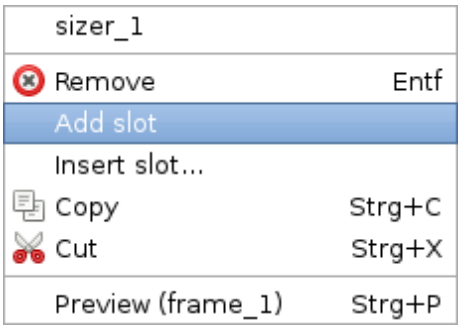


Figure 5.4: The menu for a sizer

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.

5.3 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.

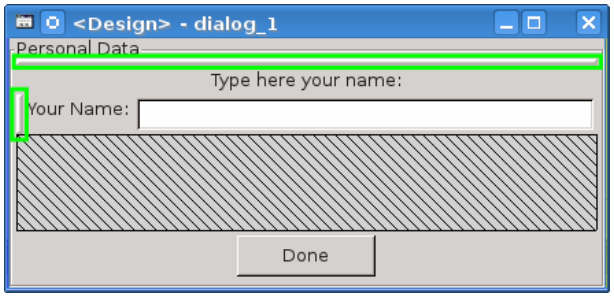


Figure 5.5: The Design Window

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”. They are marked with green boxes in the screenshot above. Click to the handles 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 5.3, “The menu for a widget” or in Figure 5.4, “The menu for a sizer”.

5.4 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.

5.4.1 Application Properties

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

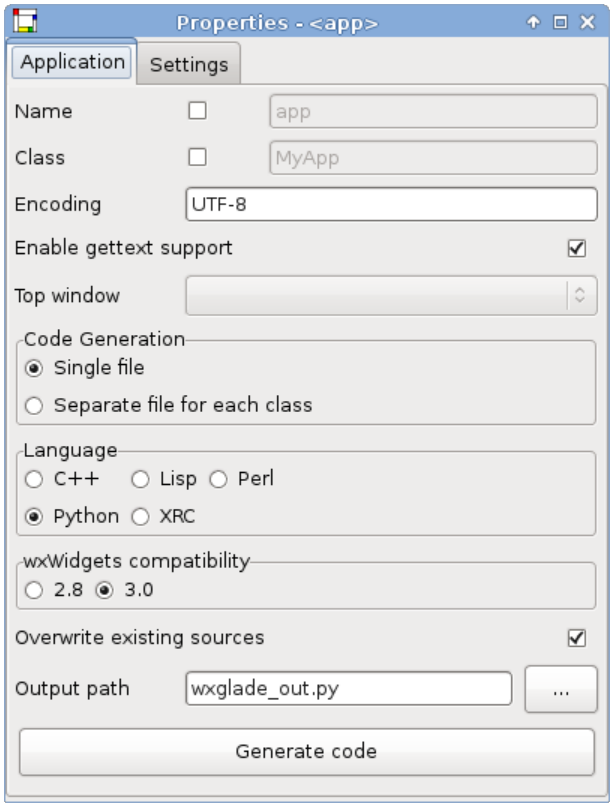


Figure 5.6: Project Properties - Application settings

“Name”

Name of the instance created from “Class”

Section 4.2.4, “Automatically Created `wxApp` Instances” provides more information

“Class”

Name of the automatically generated class derived from `wxApp`

Section 4.2.4, “Automatically Created `wxApp` Instances” 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 4.2.4, “Automatically Created `wxApp` Instances” provides more information

“Top window”

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

Section 4.2.4, “Automatically Created `wxApp` Instances” provides more information

“Code Generation”

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

Section 4.2.4, “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 4.2.4, “Using the Source Code” provides more information

Warning
This feature is deprecated and will be removed from a future version of wxGlade. Thereby it’s advised not to use this feature for new projects. Details about all deprecated features are shown in Section 3.5, “Deprecated Features”.

“Output path”

Output file or directory

Section 4.2.4, “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.

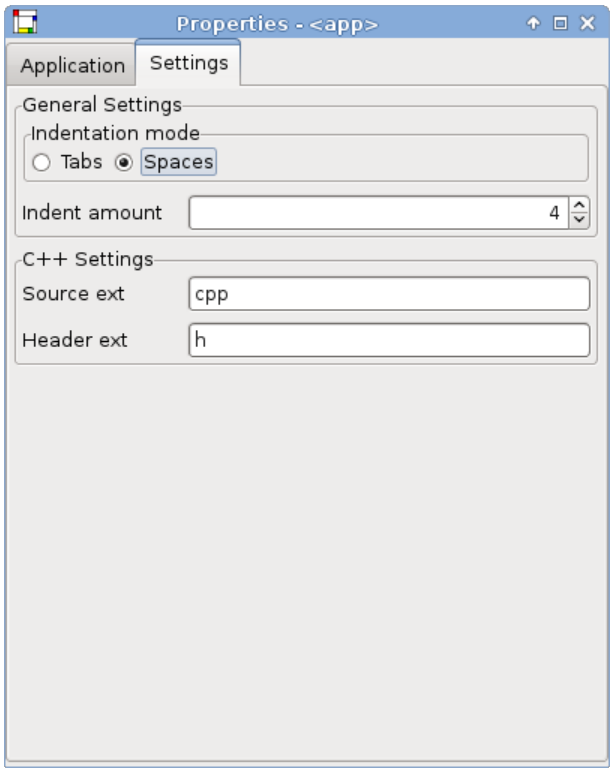


Figure 5.7: Project Properties - Language settings

“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 has a leading dot.

“Header ext”

Extension of the header file.

The extension doesn’t has a leading dot.

5.4.2 Common Properties

The first tab contains the common properties that apply to all widgets. As shown in Figure 5.8, “Common Properties” the common properties are related to name, class, size, colors, fonts and tooltip.

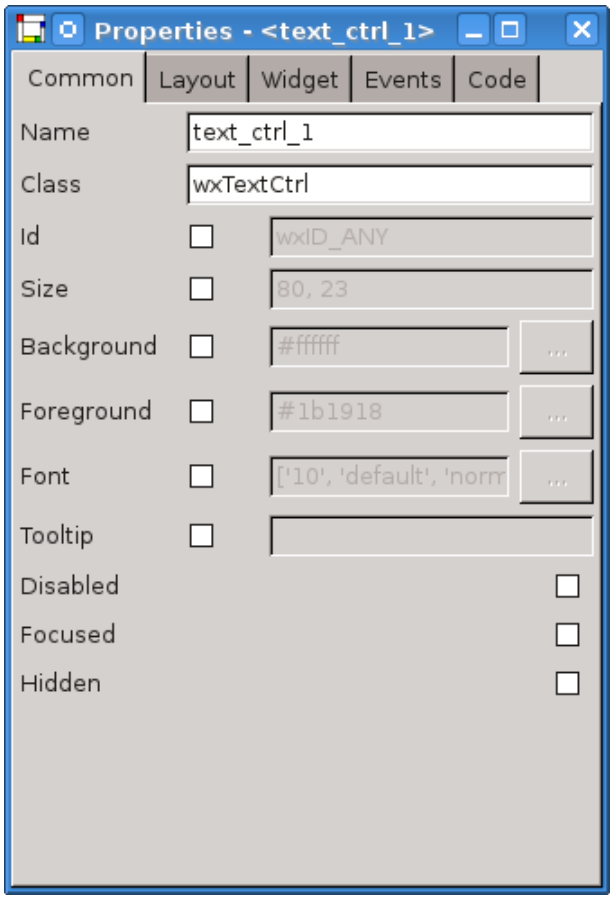


Figure 5.8: Common Properties

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 5.9, "Changing Common Properties").

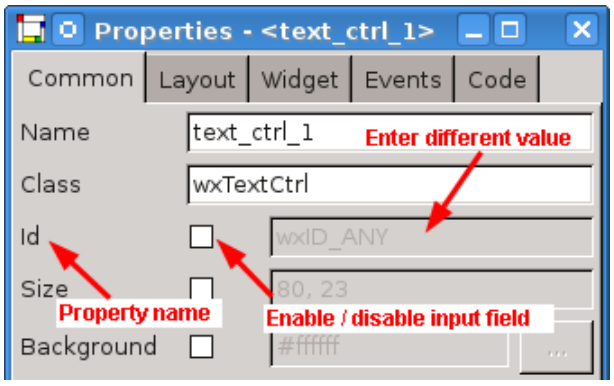


Figure 5.9: Changing Common Properties

"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.

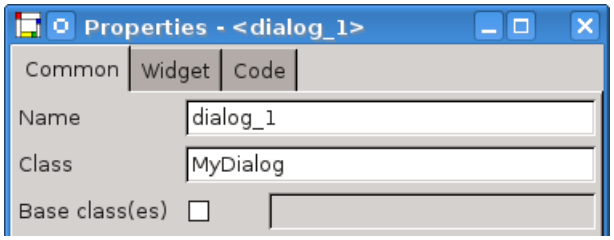


Figure 5.10: Common Properties - A subclassed widget (default behaviour)

Example 5.1 Generated Python code of a subclassed widget

```
class MyDialog(wxDialog):

    def __init__(self, *args, **kwds):
        kwds["style"] = wxDEFAULT_DIALOG_STYLE
        wxDialog.__init__(self, *args, **kwds)
```

“Base class(es)”

A comma-separated list of custom base classes. The first class 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.

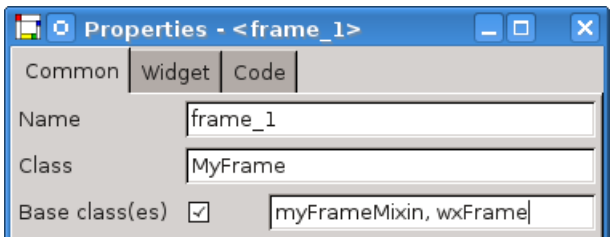


Figure 5.11: Common Properties - Base class(es) entry

Example 5.2 Generated Python code of a widget with two base classes

```
class MyFrame(myFrameMixin, wxFrame):

    def __init__(self, *args, **kwds):
        kwds["style"] = wx.DEFAULT_FRAME_STYLE
        myFrameMixin.__init__(self, *args, **kwds)
        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()**”

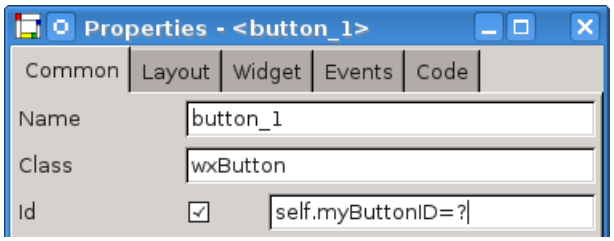


Figure 5.12: Common Properties - Variable assignment

Example 5.3 Generated Python code for a variable assignment

```
class MyFrame(wx.Frame):

    def __init__(self, *args, **kwds):
        # begin wxGlade: MyFrame.__init__
        kwds["style"] = wx.DEFAULT_FRAME_STYLE
        wx.Frame.__init__(self, *args, **kwds)
        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”

Set the widget to receive keyboard input.

“Hidden”

Hide the widget.

5.4.3 Layout Properties

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

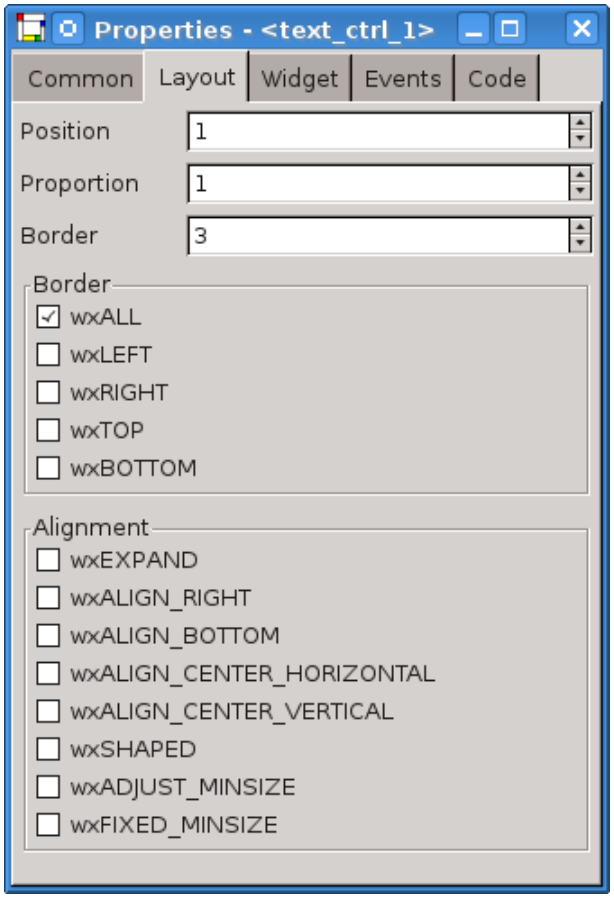


Figure 5.13: Layout Properties

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 3.4.1, "Preferences Dialog"). If you let a widget have a default border, the `wxAll` option is also checked.

5.4.4 Widget Properties

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

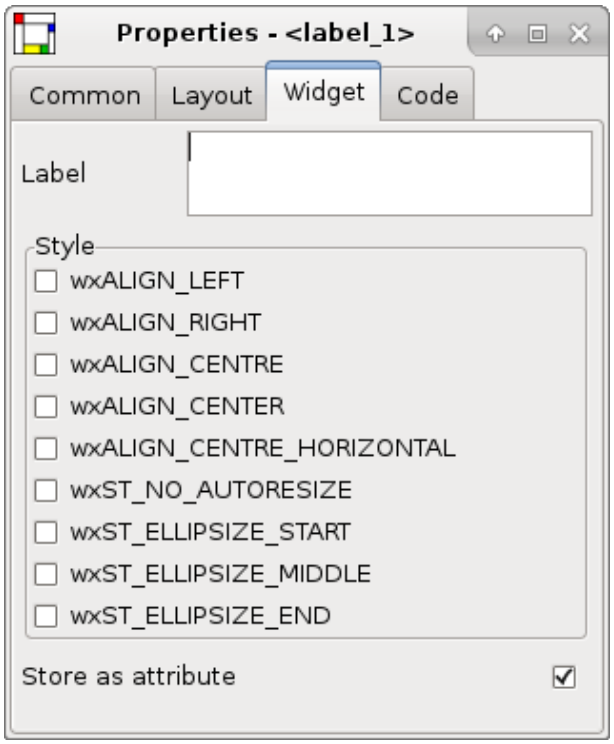


Figure 5.14: Widget Properties

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.

“Store as attribute” stores the widget as an attribute of the top-level class.

Select “Don’t generate code for this class” to prevent generating code for subwidgets of `wxNotebook`, `wxPanel` and `wxSplitterWindow`.

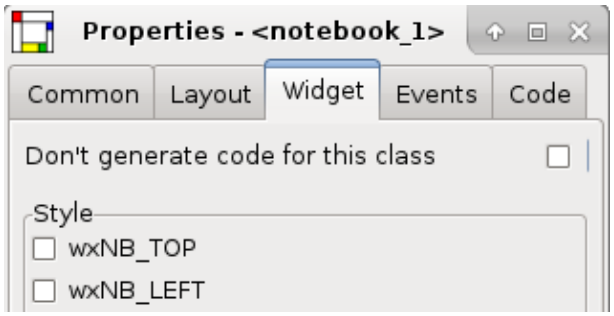


Figure 5.15: Widget Properties - Don’t generate code for this class

You can use this feature to:

1. Split complex designs into smaller pieces. The correlation in such a setup based on the widget name.
2. Prevent generation of stubs, if using custom base classes.

Some widgets have an “Apply” button onhe “Widget” tabs. Click on the “Apply” button to consolidate your changes. Otherwise they will lost.

5.4.5 Styles

The most of the styles as shown in the Section 5.4.3, “Layout Properties” and Section 5.4, “Properties Window” have a tooltip with additional information about the style.

The tooltips show a short description as well as additional information.

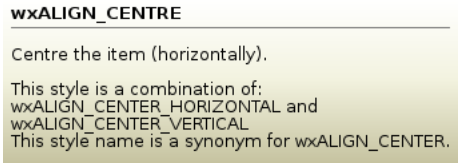


Figure 5.16: Widget Properties - Styles Tooltip

The style details will be used also to compute the styles in the generated source code.

5.4.6 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.

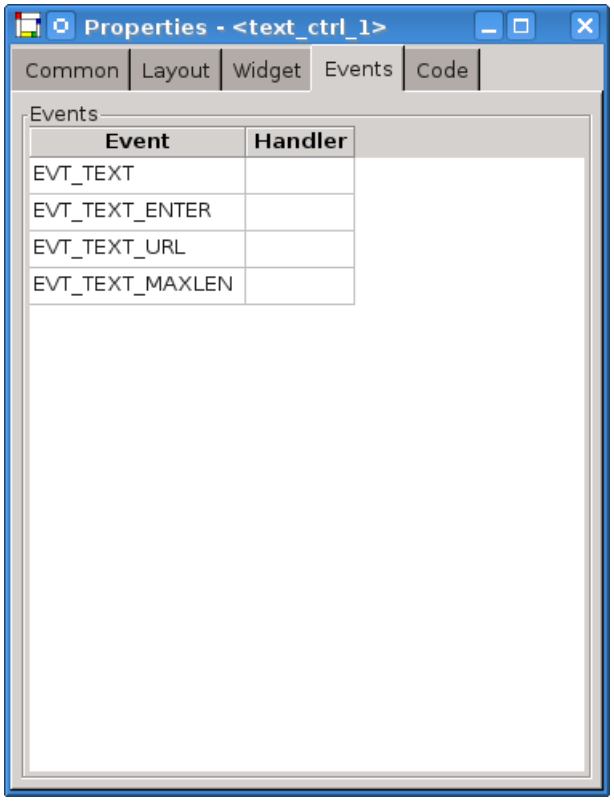


Figure 5.17: Events Properties

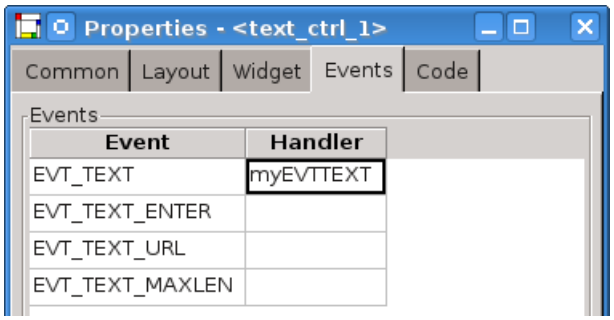


Figure 5.18: Events Properties - Event handler name added

Example 5.4 Generated Python code of an **EVT_TEXT** event handler stub at line 12

```
1  class MyFrame(wx.Frame):
2
3      def __init__(self, *args, **kwds):
4          # begin wxGlade: MyFrame.__init__
5          kwds["style"] = wx.DEFAULT_FRAME_STYLE
6          wx.Frame.__init__(self, *args, **kwds)
7          self.text_ctrl_1 = wx.TextCtrl(self, -1, "")
8          self.__set_properties()
9          self.__do_layout()
10         self.Bind(wx.EVT_TEXT, self.myEVTTEXT, self.text_ctrl_1)
11         # end wxGlade
12
13     def myEVTTEXT(self, event):  # wxGlade: MyFrame.<event_handler>
14         print "Event handler `myEVTTEXT' not implemented!"
15         event.Skip()
```

5.4.7 Code Properties

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

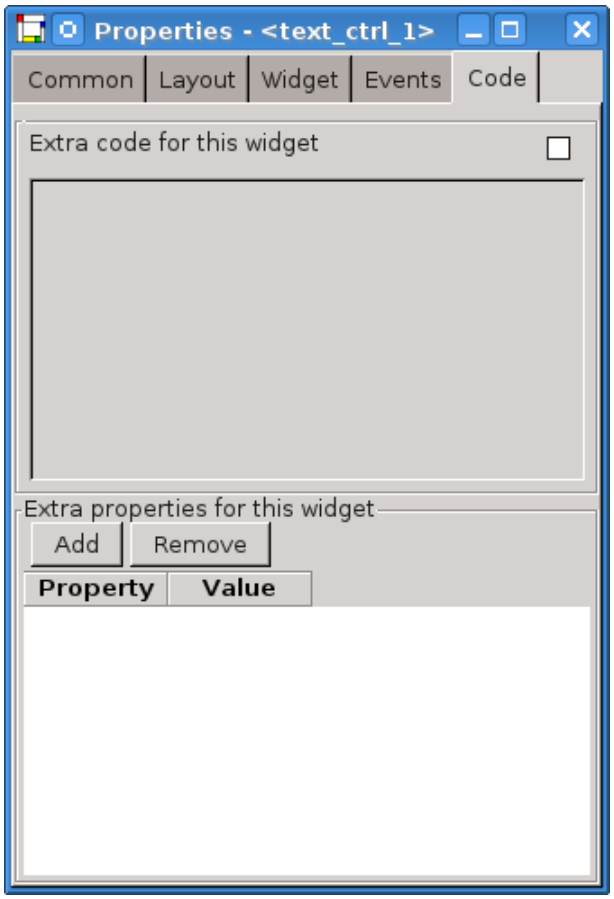


Figure 5.19: Code Properties - Extra code and extra properties

wxGlade never declares or assigns variable or function names or import for non-default base classes and custom widgets.

If you use non-default base classes, `var:` or `code:` statements, you have to manually assign the variables, declare used functions as well as to include resp. to import of additional code need by your widget in “Extra code for this widget”. This “Extra code” will be added to the context of the source file and not to the context of the class.

The lower 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.

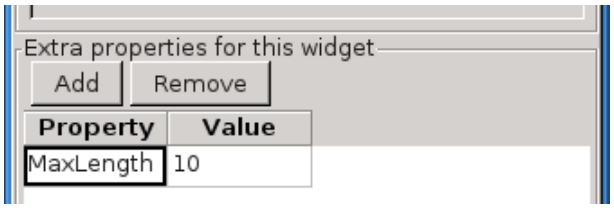


Figure 5.20: Code Properties - Set extra property

Example 5.5 Generated Python code for setting property **MaxLength** to **10** at line 14

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

5.5 The wxGlade Menu

wxGlade has only a few very small number of menus.

5.5.1 The FILE Menu

In the FILE menu 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" text box in the Preferences dialog, usually the user home directory.

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

5.5.2 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.

5.5.3 The HELP Menu

The HELP menu provides access to the wxGlade user manual (this documentation), a short tutorual and the "About..." dialog.

5.6 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 6

Supported widgets

6.1 Introduction

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

Containers	Layout Widgets	Common Widgets	Menu bar / Status Bar / Tool bar
• wxDialog • wxFrame • wxNotebook • wxPanel • wxScrolledWindow • wxSplitterWindow	• wxBoxSizer • wxStaticBoxSizer • wxGridSizer • wxFlexGridSizer • Spacer	• wxButton • wxBitmapButton • wxCalendarCtrl • wxCheckBox • wxChoice • wxComboBox • wxDatePickerCtrl • wxGauge • wxGenericCalendarCtrl • wxGrid • wxHyperlinkCtrl • wxListBox • wxListCtrl • wxPropertyGridManager • wxRadioBox • wxRadioButton • wxSlider • wxSpinButton • wxSpinCtrl • wxStaticBitmap • wxStaticLine • wxStaticText • wxTextCtrl • wxToggleButton • wxTreeCtrl	• wxMenu • wxMenuBar • Menu Bar Separators • Sub Menus • wxStatusBar • wxToolBar • Tool Bar Separators • Tools

Missing widgets can be integrated using the “Custom Widget”.

6.2 Specifying the Path of Bitmaps

You need to specify a bitmap in some widgets. This can be done in several ways:

6.2.1 Bitmap Path

Specify the path to the bitmap file.

Usually you can type an absolute path in a text box or browse for a bitmap with a file dialog.

Example 6.1 `wxBitmap` object with the typed string as bitmap path

Input:
`/usr/share/icons/application.png`
produces in C++:

```
wxBitmap("/usr/share/icons/application.png", wxBITMAP_TYPE_ANY)
```

6.2.2 statement "`var:`"

Syntax: *var:<variable name>*

You can enter a variable name using the *var:* tag in the text box. In Perl code generation a "$" sign is added if you omit it.

Example 6.2 `wxBitmap` object with the variable name as bitmap path

Input:
`var:my_bitmap_path`
produces in C++:

```
wxBitmap(my_bitmap_path, wxBITMAP_TYPE_ANY)
```

6.2.3 statement "`empty:`"

Syntax: *empty:<width>,<height>*

Creates an empty bitmap of the specified size. It's recommended to use a minimal size of 1, 1.

Example 6.3 Create an empty `wxBitmap` with width of 32 and height of 32

Input:
`empty:32,32`
produces in Python:

```
wx.EmptyBitmap(32, 32)
```

produces in C++:

```
wxBitmap(32, 32)
```

6.2.4 statement "`art:`"

Syntax: *art:<ArtID>,<ArtClient>* or *art:<ArtID>,<ArtClient>,<width>,<height>*

Create a bitmap using wxArtProvider.

Example 6.4 Create a bitmap using wxArtProvider

Input:
`art:wxART_GO_UP,wxART_OTHER,32,32`
produces in Perl:

```
Wx::ArtProvider::GetBitmap(wxART_GO_UP, wxART_OTHER, Wx::Size->new(32, 32))
```

6.2.5 statement "code:"

Syntax: *code:<code chunk to return a wxBitmap>*

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.

Example 6.5 `wxSomeWidget` needs a `wxBitmap` as an argument

Input:
`code:if (x == 0) get_bitmap1() else get_bitmap2();`
produces in C++:

```
wxSomeWidget((if (x == 0) get_bitmap1() else get_bitmap2();), option1, option2)
```

If you use the `code:` tags like shown above the preview window shows a fixed size empty bitmap instead.

Note
Refer to Section 5.4.7, "Code Properties" for a description of declaration and assignment of additional functions and variables.

6.3 Menu, Statusbar and Toolbar

6.3.1 Introduction

wxGlade helps you to design the menu, the tool bar and the status bar for your application.

There are two ways to integrate the three elements into a wxGlade project:

1. Create standalone widgets by clicking the corresponding button in the main window.
2. Associated with a `wxFrame`by selecting the related checkboxes in the `wxFrame` properties window.

Associated bars will be integrated automatically to the right place. On the other side standalone widgets have to integrate the elements later manually in design after the source code has been generated.

6.3.2 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.

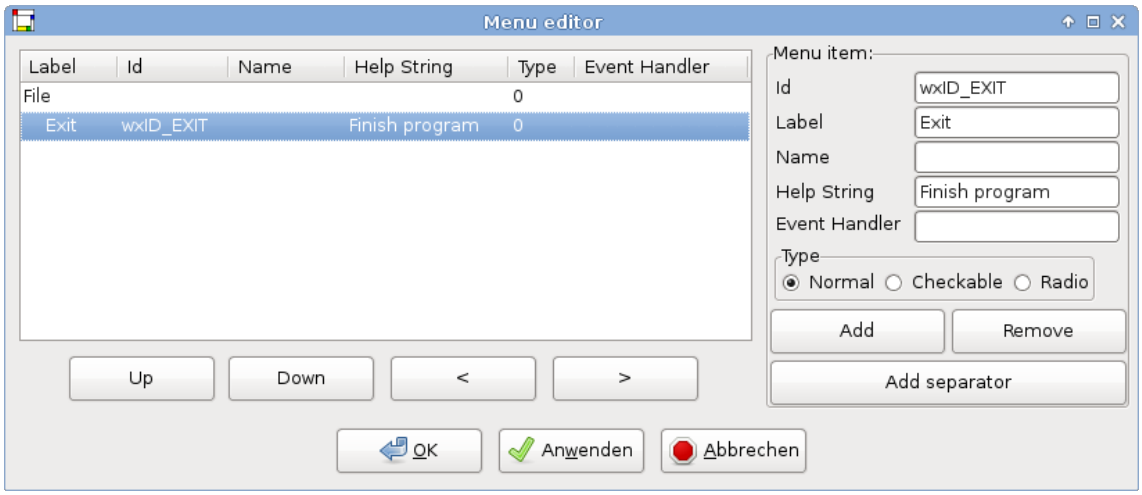


Figure 6.1: Menu editor

6.3.3 Statusbar

In the “Properties” window you can edit the list of fields and their size.

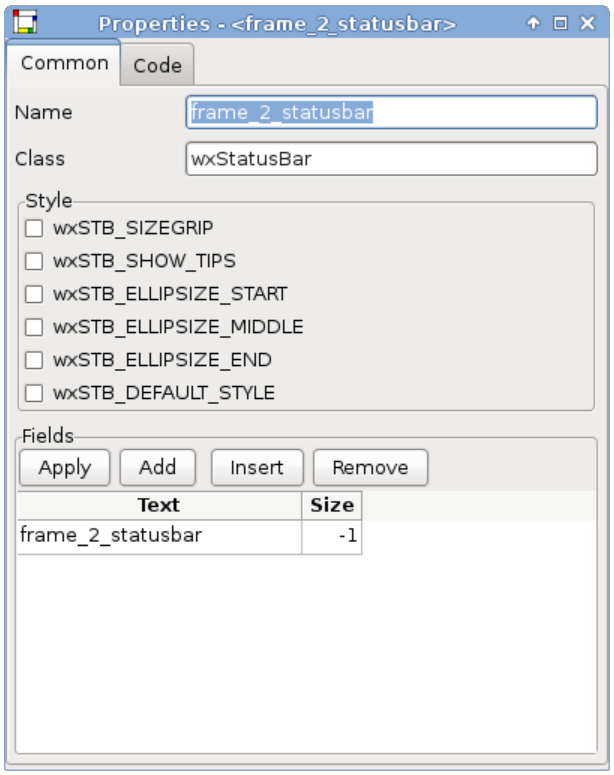


Figure 6.2: Statusbar properties

6.3.4 Toolbar

You can edit the toolbar 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 6.2, “Specifying the Path of Bitmaps” for bitmap path specifications.

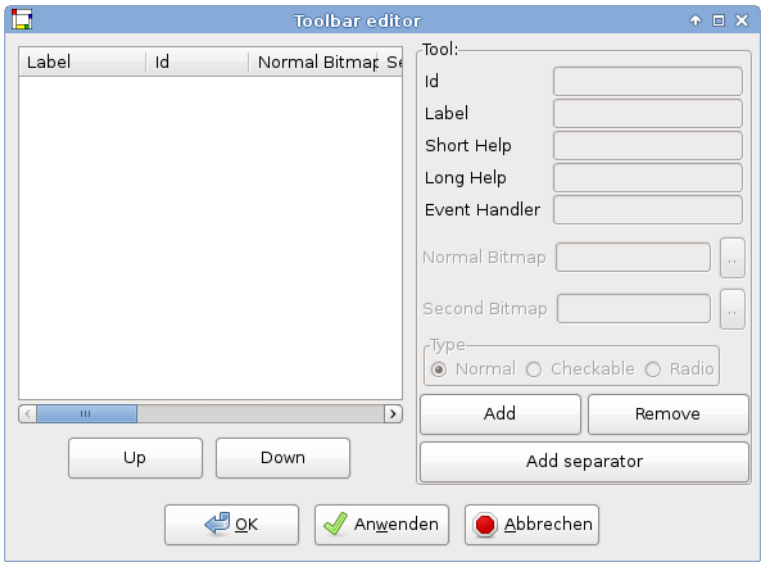


Figure 6.3: Toolbar editor

6.4 Widget List

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

6.4.1 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.

6.4.2 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.

6.4.3 Panel

This allows you to add a panel to a sizer. In the “Properties” window you can choose the styles.

6.4.4 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.

6.4.5 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.

6.4.6 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 6.4.6, "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 6.2, "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.

Spin Button

This produces a `wxSpinButton`. In the "Properties" window you can enter the range and the value.

Toggle Button

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

6.4.7 Calendar Control

This produces a `wxCalendarCtrl`. In the "Properties" window you can enter the style.

6.4.8 Check List Box

This produces a `wxCheckListBox`. In the "Properties" window you can enter the choices, the selection and the style.

6.4.9 Check Box

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

6.4.10 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.

6.4.11 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.

Click on the "Apply" button to consolidate your changes, they are lost without clicking.

6.4.12 Date Picker Control

This produces a `wxDatePickerCtrl`. In the "Properties" window you can enter the style.

6.4.13 Gauge

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

6.4.14 Generic Calendar Control

This produces a `wxGenericCalendarCtrl`. In the "Properties" window you can set the style.

6.4.15 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. Also you can choose to let wxGlade to create the grid or leave it to the user code.

6.4.16 Hyperlink Control

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

6.4.17 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.

6.4.18 List Control

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

6.4.19 Property Grid Manager

This produces a `wxPropertyGridManager`. In the "Properties" window you can set the style.

6.4.20 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.

6.4.21 Slider

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

6.4.22 Spin Control

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

6.4.23 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.

6.4.24 Static Bitmap

This produces a `wxStaticBitmap`. You will be prompted for the bitmap path. Refer to Section 6.2, "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.

6.4.25 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.

6.4.26 Text Control

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

6.4.27 Tree Control

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

6.4.28 Custom Widget

When you put a Custom Widget in the design window you will be prompted for a class name.

Note
Custom Widgets will not be shown in the design preview.

Constructor for Custom Widgets

In the “Widget” tab of the “Properties” window you can set a number of custom attributes that will appear in the constructor call.

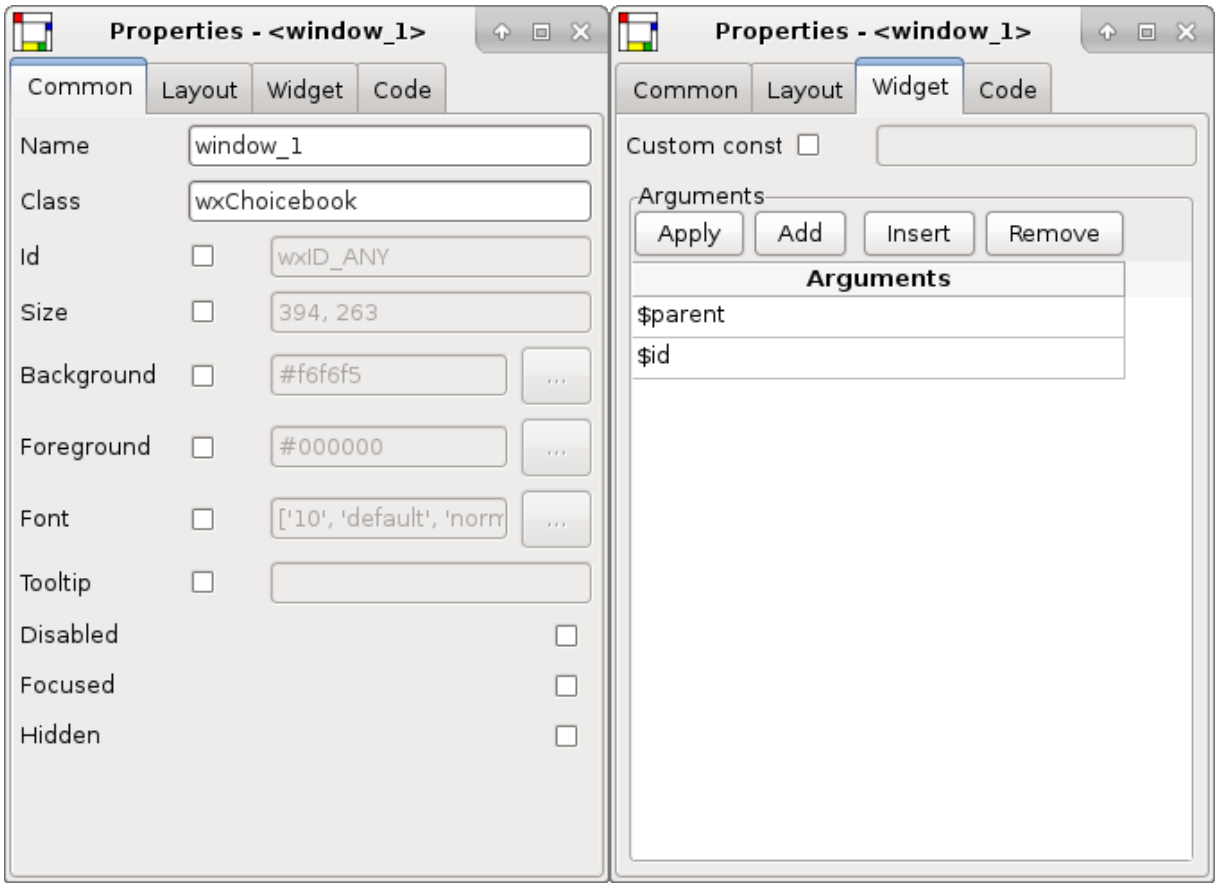


Figure 6.4: Widget Properties for a Custom Widget

There are four attributes **`$id`**, **`$parent`**, **`$width`** and **`$height`** have a special meaning:

$id
: This attribute will be replaced by the own widget ID e.g. `wxID_ANY`.

$parent
: This attribute will be replaced by a reference to the parent widget.

$width
: This attribute will be replaced by the widget width.

$height
: This attribute will be replaced by the widget height.

These attributes have different effects in C++, Lisp, Perl, Python or XRC code generation.

For XRC you can use it to add custom attributes to the resource object. To do so, arguments must have the following format: “*`ATTRIBUTE_NAME:ATTRIBUTE_VALUE`*”. For instance: “*`default_value:10`*” is translated to: “`<default_value>10</default_value>`”. Invalid entries are silently ignored.

You can use the property “Custom constructor” to specify a custom constructor like a factory method.

Example 6.6 Generated C++ code for the custom widget shown above

```
MyFrame::MyFrame(wxWindow* parent, int id, const wxString& title, const wxPoint& pos, const wxSize& size, long style):
    wxFrame(parent, id, title, pos, size, style)
{
    // begin wxGlade: MyFrame::MyFrame
    window_1 = new wxChoicebook(this, wxID_ANY);

    set_properties();
    do_layout();
    // end wxGlade
}
```

Note
Refer to Section 5.4.7, "Code Properties" for a description of declaration and assignment of additional functions and variables.

Example: AGW SpeedMeter

Example 6.7 Widget Custom Widget - AGW SpeedMeter

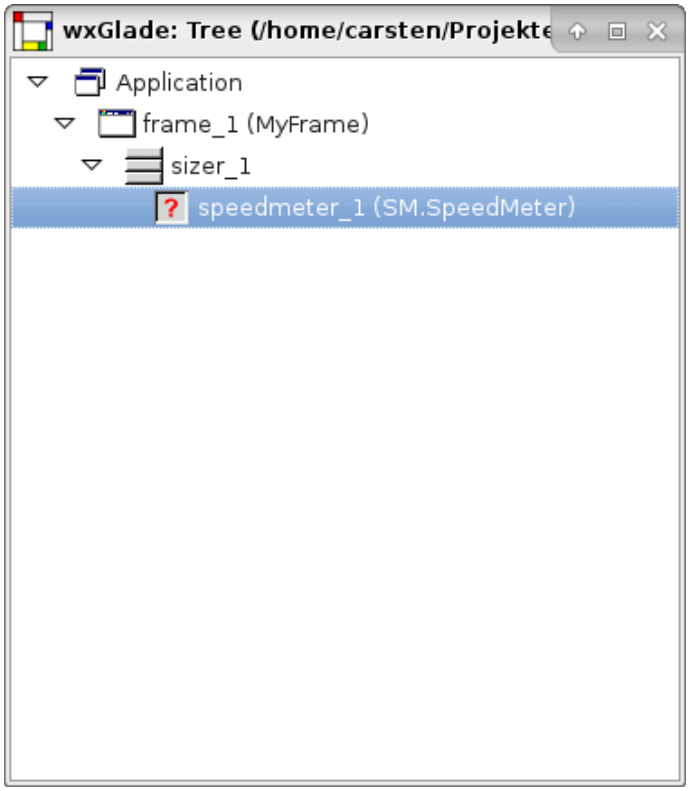


Figure 6.5: Widget Tree

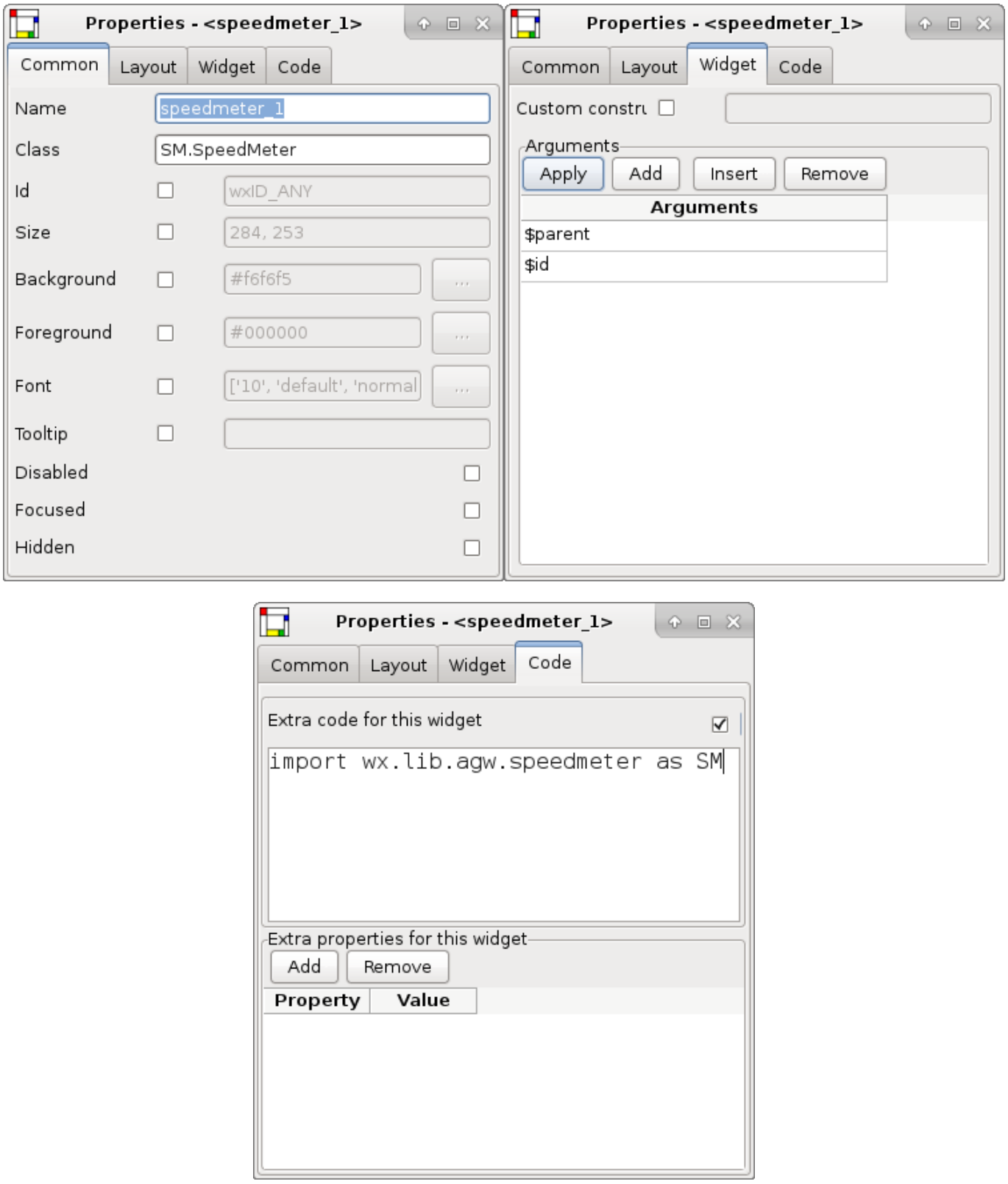


Figure 6.6: SpeedMeter Properties

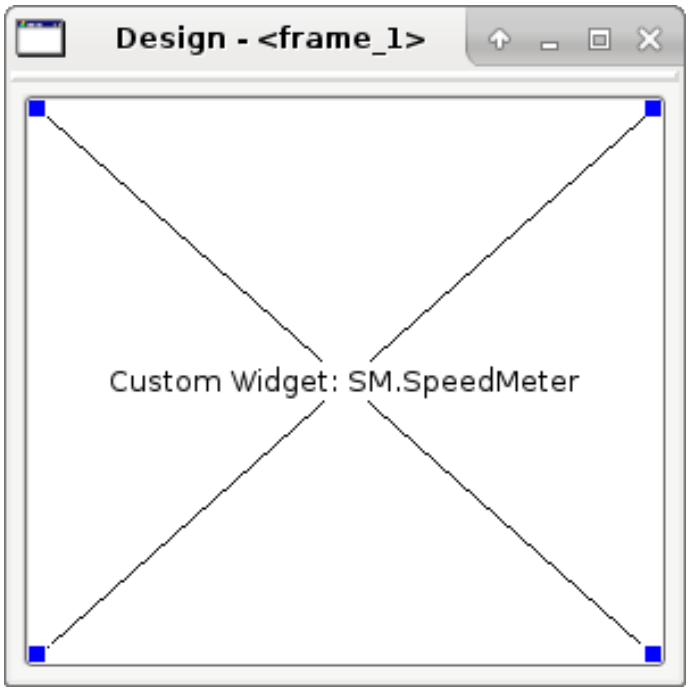


Figure 6.7: Preview

Generated Python code:

```
#!/usr/bin/env python
# -*- coding: UTF-8 -*-
#
# generated by wxGlade 0.7.1 on Sat Dec 19 11:11:39 2015
#

import wx

# begin wxGlade: dependencies
import gettext
# end wxGlade

# begin wxGlade: extracode
import wx.lib.agw.speedmeter as SM
# end wxGlade


class MyFrame(wx.Frame):
    def __init__(self, *args, **kwds):
        # begin wxGlade: MyFrame.__init__
        wx.Frame.__init__(self, *args, **kwds)
        self.speedmeter_1 = SM.SpeedMeter(self, wx.ID_ANY)

        self.__set_properties()
        self.__do_layout()
        # end wxGlade

    def __set_properties(self):
        # begin wxGlade: MyFrame.__set_properties
        self.SetTitle(_("frame_1"))
        self.SetSize((300, 300))
        self.speedmeter_1.SetSpeedValue(33)
        # end wxGlade
```

```
34
35      def __do_layout(self):
36          # begin wxGlade: MyFrame.__do_layout
37          sizer_1 = wx.BoxSizer(wx.VERTICAL)
38          sizer_1.Add(self.speedmeter_1, 1, wx.ALL | wx.EXPAND, 5)
39          self.SetSizer(sizer_1)
40          self.Layout()
41          # end wxGlade
42
43  # end of class MyFrame
44  class MyApp(wx.App):
45      def OnInit(self):
46          frame_1 = MyFrame(None, wx.ID_ANY, "")
47          self.SetTopWindow(frame_1)
48          frame_1.Show()
49          return True
50
51  # end of class MyApp
52
53  if __name__ == "__main__":
54      gettext.install("app") # replace with the appropriate catalog name
55
56      app = MyApp(0)
```

6.4.29 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 7

wxGlade technical notes

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.

7.1 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.

7.1.1 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
```

7.1.2 Create a 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
```

7.1.3 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 3.4.1, "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 exists.

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

7.1.4 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 file `wconfig.py` in the widget directory and describe the styles used by this widget

```
"""\
wxStaticLine widget configuration

@copyright: <Add year and your name>
@license: <Choice a license>
"""

config = {
    'wxklass': 'myCtrl',
    'style_defs': {
        'wxMCFance': {
            'desc': _('Use the new and fancy design.'),
        },
        'wxMCOldFashion': {
            'desc': _('Use the old fashion design.'),
        },
    },
    'box_label': _('Style'),
    'default_style': 'wxMCFance',
    'style_list': ['wxMCFance', 'wxMCOldFashion']
}
```

4. Create a Python file codegen.py with initial content like

```
"""
Code generator functions for myCtrl objects

@copyright: <Add year and your name>
@license: <Choice a license>
"""

import common


class PythonMyCtrlGenerator(wcodegen.PythonWidgetCodeWriter):

    tmpl = '%(name)s = %(klass)s(%(parent)s, %(id)s, %(label)s%(style)s)\n'

# end of class PythonMyCtrlGenerator


def initialize():
    common.class_names['EditmyCtrl'] = 'myCtrl'

    pygen = common.code_writers.get("python")
    if pygen:
        pygen.add_widget_handler('myCtrl', PythonMyCtrlGenerator())
```

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

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

Widget Initialisation

Note
This section is incomplete.

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()`)

7.2 Contributing to wxGlade

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.

7.2.1 Coding conventions

- Class names are usually CamelCase - variables, functions and method names are lower_case_with_unserscores
- 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

7.2.2 Testing

Please write tests for significant changes and run the test suite before committing changes.

7.2.3 Commit Messages

- Use the present tense ("Add feature" not "Added feature")
- Use the imperative mood ("Move cursor to..." not "Moves cursor to...")
- Limit the first line to 72 characters or less
- Reference issues and pull requests liberally

7.3 Incomplete and outdated parts

Caution
The remaining content in this section has not been updated since July 2002. It's likely be very outdated in some parts.

7.3.1 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/" subdirectory 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' dictionary (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

7.3.2 Adding a top-level Widget

When the user clicks on a button of a top-level 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 application. 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 top-level windows) and the zero-based position inside the sizer (once again, this is unused for top-level 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' dictionary

7.3.3 Adding a top-level Sizer

This is similar to the addition of a top-level 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 top-level 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

7.3.4 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 "on_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-top-level 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.

7.3.5 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

7.3.6 Saving the Design

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.

7.3.7 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

7.3.8 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 top-level 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.

Appendix A

Glossary of Terms, Abbreviations, and Acronyms

The following abbreviations are used in this manual:

API Application Programming Interface

It's a standardized implementation-independent programming interface.

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 Numeronym for internationalisation support.

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

Managed Widget Normal widget with a parent object

Non-Managed Widget Synonym for lop-level widget

OS Operating system

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

Python Python is a general-purpose, high-level programming language.

SAE Standalone Edition

Top-level widget Top-level widgets like `wxFrame` or `wxDialog` don't have a parent object.

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.

UTF-8 Universal Character Set Transformation Format - 8-bit

Character encoding with the capability to encode all possible characters

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

wx abbreviation for wxWidgets

wxPython is a wrapper for wxWidgets of the Python programming language.

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, Licenses and Trademarks

B.1 Copyrights

wxGlade is copyright 2002-2007 by Alberto Griggio and 2011-2016 by Carsten Grohmann. Use and distribution of wxGlade is governed by the MIT license, located in Section B.2, “wxGlade License Agreement”.

B.2 wxGlade License Agreement

Copyright (c) 2002-2007 Alberto Griggio.

Copyright (c) 2011-2016 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.

B.3 Trademarks

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.

B.4 Licenses and Acknowledgements for Incorporated Software

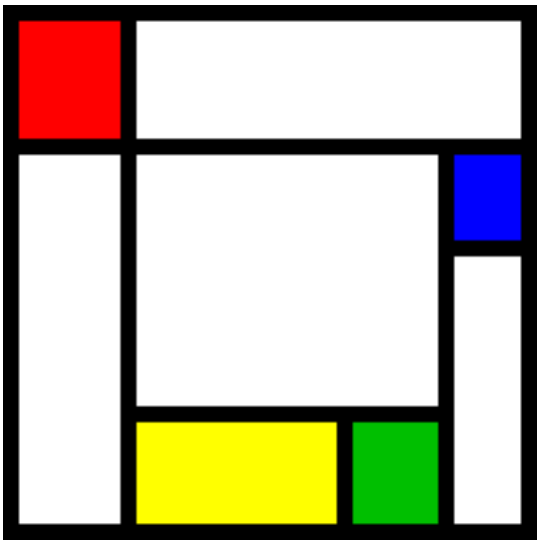
This section lists licenses and acknowledgements for third-party software incorporated in wxGlade.

B.4.1 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:

Appendix C

The wxGlade Icon



The wxGlade logo is a homage to Piet Mondrian a Dutch painter.

The icon looks like a typical Mondrian painting. It has the characteristic solid horizontal and vertical black lines as well as areas filled with the three primary colours yellow, red and blue.