

Programming With wxDev-C++

**Covering using wxDev-C++ for
Programming and Debugging**

Basic C and C++ Programming

**Using wxWidgets with wxDev-
C++**

**Answers Frequently Asked
Questions**

Source Code Available Online

By Sof.T

Copyright (C) 2006 Sof.T

This book and associated source code is free published material; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This book and associated source code is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this book; if not, write to the Free Software Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.

Version No: 0.7.1

Release Date: 17th October 2007

Newest version available from

http://sourceforge.net/project/showfiles.php?group_id=173102

Programming With wxDev-C++

“It’s got some quirks but then again don’t we all”

NinjaNL

Contents

The Boring Bit

- Introduction
- History of wxDev-C++
- Who This Book Is For
- Acknowledgements

Part 1 – C / C++ Programming with wxDev-C++

- **Chapter 1 – Downloading, Installing and Updating wxDev-C++**
 - Introduction
 - Downloading wxDev-C++
 - Installing wxDev-C++
 - Updating wxDev-C++
 - Adding Extra Packages
- **Chapter 2 – Compiling your first program**
 - Introduction
 - Opening an existing project
 - Creating your own project
- **Chapter 3 – Basic C Programming**
 - Introduction
 - Break down of a simple example
 - Basic C
 - Conditional Loops
 - Conditional Execution
 - Preprocessor
 - Functions
 - Input/Output and other useful functions
 - Pointers
 - Memory allocation
 - Arrays
 - Strings
 - Structures
- **Chapter 4 – Basic C++ Programming**
 - Introduction
 - Break down of a simple example
 - Basic C++
 - Functions
 - Memory allocation

- Classes
- Input/Output
- Strings
- Namespaces
- Exceptions
- Templates
- **Chapter 5 – Finding your way around the IDE**
- **Chapter 6 – Debugging with wxDev-C++**
- **DevC++ Related FAQs**
 - What is this I hear about an Easter egg in Dev-C++?
 - Why is my program not recompiled when one of the header files is altered?

Part 2 – Basic Development with wxWidgets and wxDev-C++

- **Chapter 7 – Creating a Basic HTML Editor**
 - Introduction
 - Starting a wxWidgets Project
 - Using the Form Designer
 - Altering Properties
 - Adding Code
- **Chapter 8 – Working With Frames and Dialogs**
 - Introduction
 - Creating a New Project
 - Multiple Frame Project
 - Adding a New Form
 - Adding a New Dialog
 - Sample Application Part 1 – The outline
- **Chapter 9 – The Component Palette**
 - Introduction
 - Adding Components to a Frame or Dialog
 - Altering Components
 - Direct Manipulation of Components
 - The Designer Form Context Menu
 - Component Properties
 - The Components
 - Sizers
 - Controls
 - *Text Controls*
 - *Buttons*
 - *Choices*
 - *The Rest*
 - Window
 - Toolbar
 - Menu
 - Dialogs
 - System
 - MMedia

- Unofficial
 - Sample Application Part 2 – The GUI
 - The Main Form
 - The About Box
 - The Splash Screen
 - The Insert Hyperlink Dialog
 - The Create Table Dialog
 - The Insert Image Dialog
 - Advanced Users
- **Chapter 10 – Making It Work With Code**
 - Introduction
 - Where to add your code (or where did it go)
 - Responding to Events
 - Types of Event
 - Sample Application Part 3 – The code
 - The Application Code
 - The Splash Screen
 - The Dialogs
 - The Main Frame
 - Making it all work
 - Tidying Up
- **Chapter 11 – Guidelines for Professional Looking Programs**
 - Naming Conventions
 - Mnemonics or Keyboard Accelerators
 - Working with Multiple Source Code Files
 - Other Points
 - Sample Application Part 4 – Making it professional
- **Basic wxDev-C++ Related FAQs**
 - Why does my program look different when I compile it?
 - Why does my program not look like a Windows XP program when I compile it?
 - wxDev-C++ keeps crashing what's wrong, what can I do?
 - I started wxDev-C++ then selected File|New|New wxFrame but when I pressed compile nothing happens.
 - I try to compile my project but get lots of errors.
 - I disabled code completion, but the visual designer keeps complaining, why does the designer need code completion enabled?
 - I have installed a new version of wxDev-C++ or a new version of the wxWidgets library. Now when I try to compile a previously working project I get this error “cannot find -lwxmsw25”. What is wrong?
 - I have installed a new version of wxDev-C++ or a new version of the wxWidgets library. Everything compiles fine, but when I try to run the program I get a library mismatch error dialog. What is wrong?
 - I altered a line in the wxWidgets setup.h file to enable a feature I need. Now when I build my program I get a library mismatch dialog. What did I do wrong?

- I try to compile my wxWidgets project but I keep getting the error “[Resource error] can’t open cursor file ‘wx/msw/hand.cur’: No such file or directory. What am I doing wrong?”
- I moved my project files (or downloaded the source code examples) and now I’m getting a resource error.
- I am using wxDev-C++ Version 6.10 or 6.10.1 and the class browser does not seem to be working. What can I do?
- I am using wxDev-C++ with the Microsoft compiler. At link time I get the error <filename>.obj : error LINK2001 : unresolved external symbol __pexit. What is wrong?
- I want to create large frames but I cannot drag the frame beyond the boundaries of the designer or access all of the frame.
- I have one or more floating yellow boxes containing function definitions that reappear every time I load my project. Even when I minimise wxDev-C++ they are still there. How can I get rid of them?
- Where can I go for more help?

Part 3 – Advanced Development with wxWidgets and wxDevC++

- **Chapter 12 - Creating and using other controls**
 - The Simple Way
 - The Complex Way
- **Chapter 13 – Working with other frameworks**
 - OpenGL
 - SDL

Part 4 – Going Beyond the Boundaries of this Book

- Alternatives
- Resources

Appendix

- A. Keyboard Shortcuts**
- B. C/C++ Keywords**
- C. Appendix C – wxDev-C++ event names and wxWidgets equivalents**
- D. C Standard Libraries**
- E. C++ Standard Libraries**

This Page Intentionally Left Blank
(Just to irritate you when you print it out)

The Boring Bit

Introduction

The first question any child will ask you is *why* and it is a good question (except after the thirtieth time of being asked). The sun is blazing down; it's a 100⁰ outside, so why am I indoors writing this book?

The main reason is because many people have asked on wxForum if a book on wxDev-C++ is going to be written. So far a few tutorials have been produced and there have been various mutterings about books. That answers the question as to why I am writing **this book**. But not why **I** am writing this book.

For me wxDev-C++ is something very special. It all goes back to July 1999 (queue the flashback and misty camera lenses). As per usual I brought a computer magazine, just one in a huge pile of magazines, but this one was special. On the cover disc was Championship Manager 3, but being a geek I was not interested in this. Rather I was taken by the small box in the corner saying 'Bloodshed DevC++, Free C and C++ environment'. A whole new world was opened to me, the world of C and C++ up to then I had only programmed in Basic and Visual Basic. I was also introduced to the amazing world of Open Source Software.

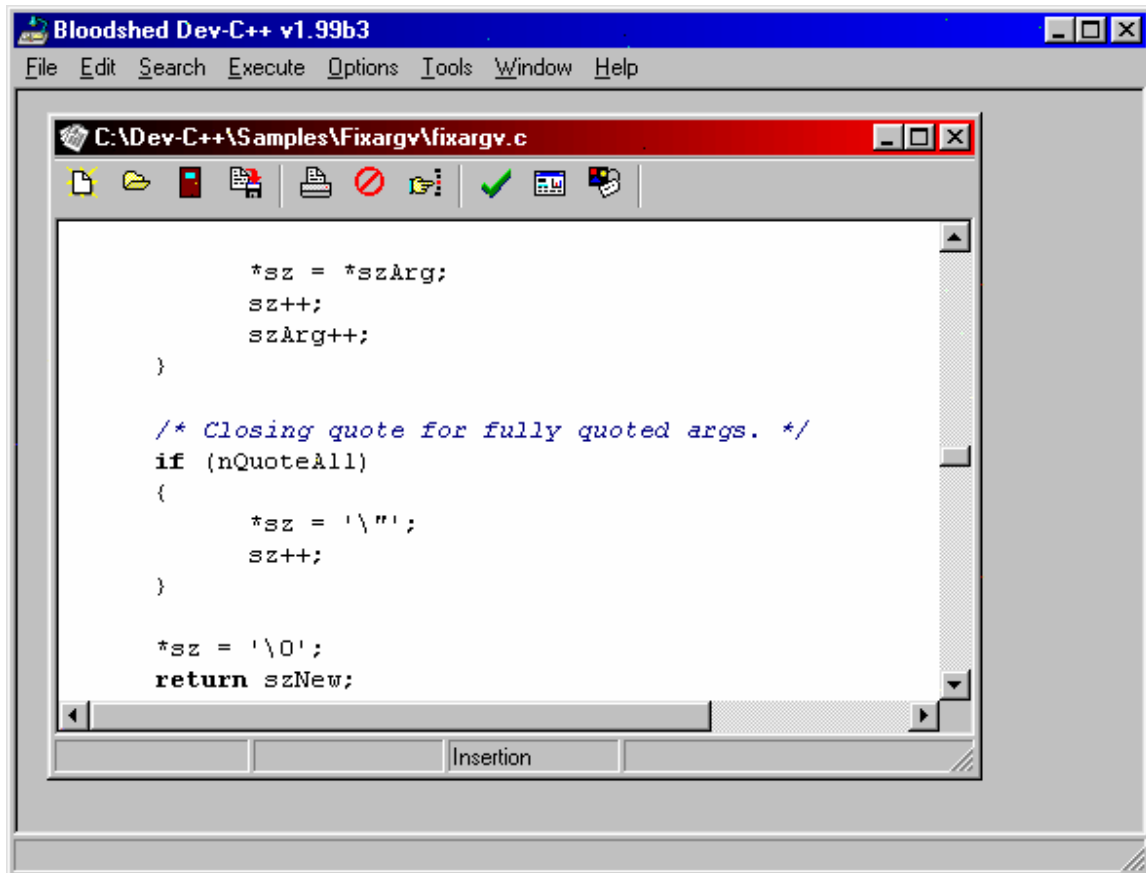


Figure 0.1 – Dev-C++ circa 1999

I rushed home from work and installed this program, it was very basic and rather ugly, but I didn't notice. I opened one of the samples, pressed compile and got greeted by a message that this program could not be compiled, then the IDE (Integrated Development Environment) crashed. Undeterred I reloaded DevC++ and another sample and this time it compiled. It was just a basic window with a button in it, but to me it was a miracle. I thought Colin Laplace godlike to produce this program for free and give it away. In the years that followed I continued to use DevC++ and watched it grow from an ugly, unstable program, to an IDE reminiscent of Microsoft Visual Studio which fulfilled most of my programming needs. I am not alone, today DevC++ is still the most download development application on SourceForge.

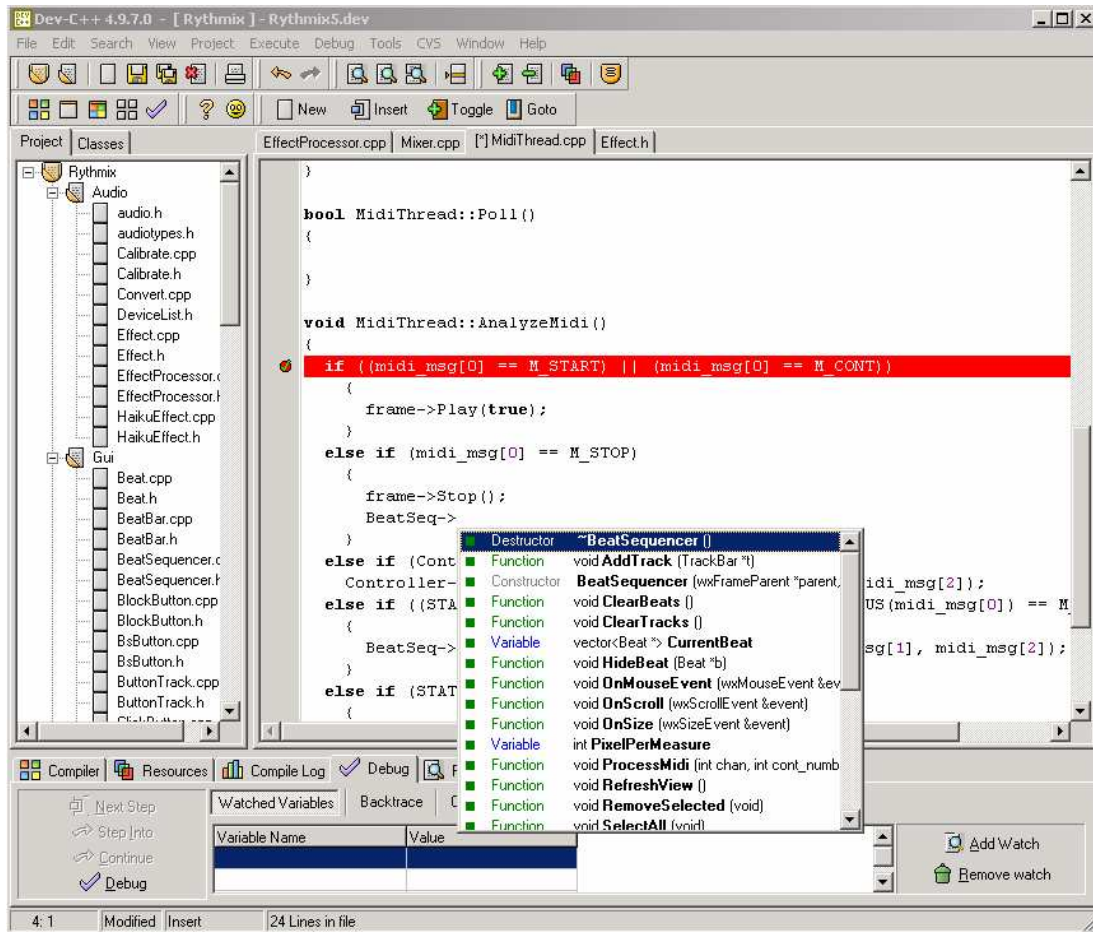


Figure 0.2 – DevC++ in a more recent guise

However I also used another IDE this time from Borland called C++ Builder. I loved the ease of use in creating GUIs in seconds. I could see what they would look like before I compiled and I could alter things in seconds that would take lines of code to create and change. I was torn between DevC++ and C++ Builder until I joined a project called 'SkinDoc' on SourceForge in 2005. This project was headed by a programmer known as Nuklear Zelph and was being developed using an application which had slipped past me called wxDev-C++. Basically it was a visual designer built on top of DevC++. I downloaded it and found the answer to all my programming desires.

Well not quite, wxDev-C++ is a great piece of work; many skilled programmers have spent and continue to spend a lot of time in creating and improving it. But it is also a work in progress, as a result it has a few rough edges. It is also similar to and yet sufficiently different to other IDEs that some parts of it maybe tricky for newcomers to use. I floundered around until I discovered the tutorials available on the wxDev-C++ home site and later discovered wxForum. wxDev-C++ also suffers from a major drawback. The original DevC++ was a paradox in that it was a C/C++ IDE written in Delphi Pascal. wxDev-C++ continues this tradition and as such programs developed with the form designer don't always match the compiled program. It is an example of a

WYSINAWYG (What You See Is Not Always What You Get) application, some of this will improve with time, but I doubt it will ever be a perfect match.

This book then is written with the experience I have gained, the experience of other users on the forums and in the hope that it will be useful. Some of it will no doubt be out of date quite soon as wxDev-C++ continues to improve, but this book will reflect these changes as it grows alongside the IDE.

Sof.T

History of wxDev-C++

In 1983 Richard Stallman announced the GNU project. This project aimed to provide a 'free' unix operating system with comparable tools. The well known GCC compiler was created by Richard Stallman as part of this project.

In 1992 Julian Smart began a project called wxWindows, which in 2004 changed its name to wxWidgets due to pressure from Microsoft. This project was designed to produce an Open Source cross platform GUI library, which used each platform's native widgets.

In 1995 Steve Chamberlain began the Cygwin project after noting that GCC contained a large number of features which made it practical to port to Windows. Other programmers joined the project and helped expand the Cygwin package.

Originally a fork from Cygwin came the MinGW (Minimalist GNU for Windows) project. This provided the tools and Windows headers for development on the Win32 platform.

Around 1999 Colin Laplace released the first version of Dev-C++. Using the MinGW compiler, it provided a minimal IDE. As other developers joined in, they helped expand Dev-C++ into an IDE which now resembles Microsoft's Visual Studio.

2003 Guru Kathiresan created a stand alone visual form designer in Delphi. Although it had limited functionality it was able to create basic applications.

In 2004 Guru Kathiresan incorporated the visual form designer into Dev-C++. The resulting application was renamed wxDev-C++ and became a RAD tool similar to Delphi or Visual Basic. Many other developers have joined in and this tool continues to improve.

Who This Book Is For

In order for this book to work it will need to address a wide range of audiences. From those who have never programmed in C++ to those who are competent, but have never

used wxWidgets, and those who are comfortable with both, but looking for an extra nugget of information.

This diverse range of prospective readers has influenced the shape of this book, experts will not want to wade through a basic primer on C/C++ programming, and beginners will not want to create GUIs that do nothing because they can not create the code to back it all up. As a result the book is divided up to allow users to skip directly to the section they are interested in.

Section one

This section deals with installing wxDev-C++, coding in C and C++, and the DevC++ part of wxDev-C++.

Section two

This section deals with GUI creation using wxDev-C++. It goes into wxWidgets and how the two work together.

Section three

The final section covers advanced topics, for users who want to do more than create using the standard controls.

Each section ends with a selection of FAQs related to that section.

Acknowledgements

Thanks to Peter James for volunteering to carry out the role of proof-reader. His edits and additions are greatly appreciated and have helped to considerably raise the quality of this book. Malcolm Nealon has also added some valuable improvements, as well as correcting at least one major mistake. Thanks also to Greg Newman for pointing out changes in the latest version of wxDev-C++.

Thanks also to the developers of wxDev-C++, especially Joel Low and Tony Reina for the time they have taken to respond to my questions.

Part 1

C / C++ Programming with wxDev-C++

Chapter 1 – Downloading, Installing and Updating wxDev-C++

Introduction

This chapter is aimed at all users. I have deliberately pitched the level of explanation to users who rarely or never install and uninstall programs. Advanced users may be irritated by the number of screenshots and the overly precise instructions. If this is you then just skip past this section or perhaps lightly skim it for information that is new to you.

A question that many have asked is if wxDev-C++ is available on platforms other than Windows. The short answer is no. However the good news is that any code you produce with wxDev-C++ can be compiled on other platforms. For more information check the FAQ at the end of Part 1 or visit <http://wxdsgn.sourceforge.net/faq.php> .

Downloading wxDev-C++

The wxDev-C++ project is hosted on SourceForge and this is the place to go to download the latest official version (There are other versions, but we will cover those later). So for now make sure you are connected to the Internet and open your web browser. To go to the official wxDev-C++ website enter the URL <http://wxdsgn.sourceforge.net>.

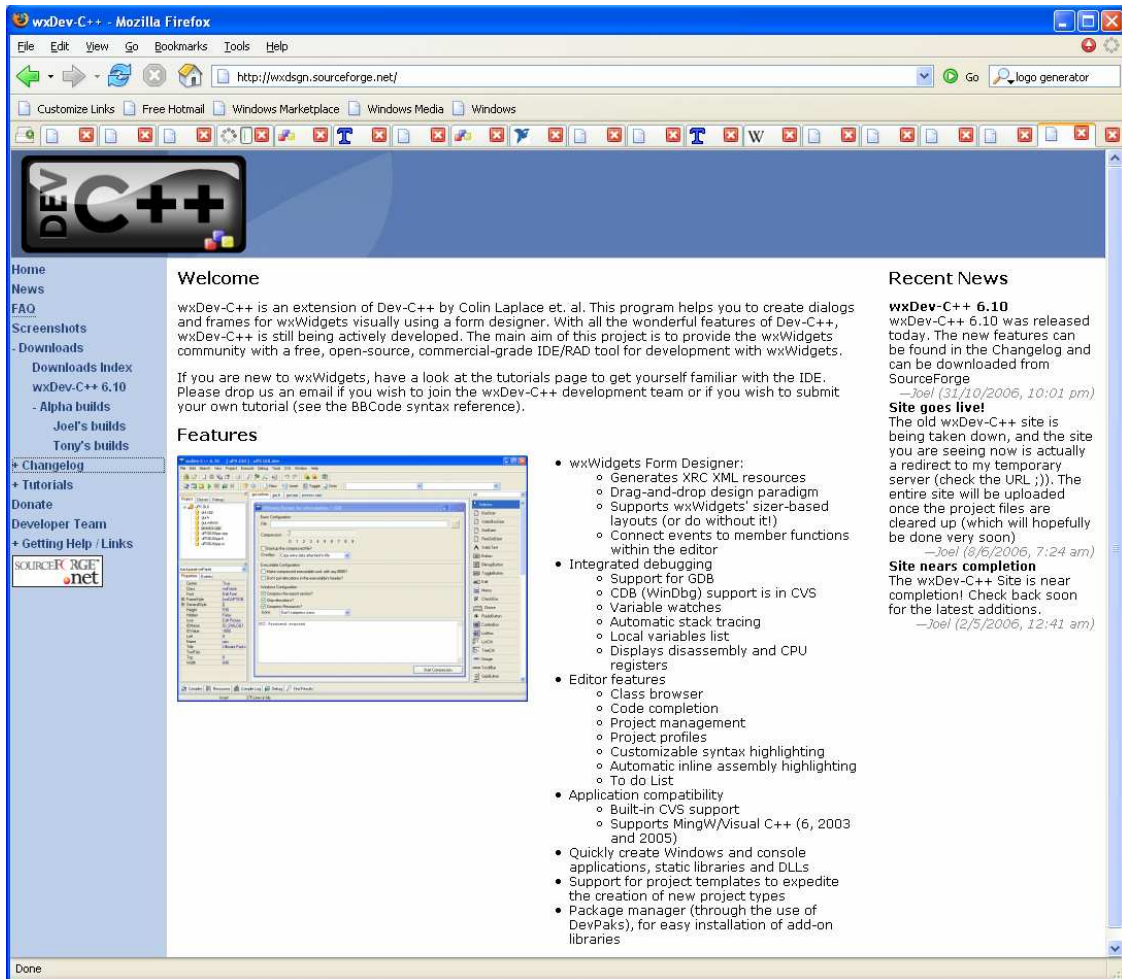


Figure 1.1 – The Official wxDev-C++ website [Accessed 1/11/2006 at 7:55A.M.]

On the navigation bar on the left you will see the link to Downloads. At present there are two different links one to wxDev-C++ and the other to wxDev-C++ for VC. The first version uses the open source compiler Mingw only but the other version can also use Microsoft's Compiler. Soon both these versions will be merged.

Click on the wxDev-C++ 6.10 option.

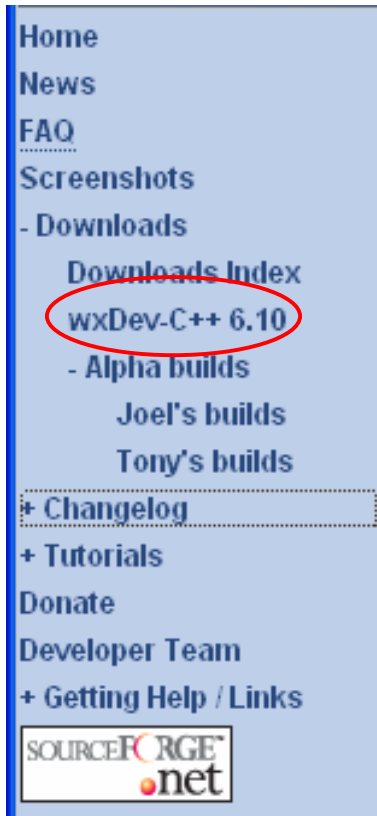


Figure 1.2 – Selecting a version of wxDev-C++

Clicking on the link labelled 'wxDev-C++ 6.10' will take you to the SourceForge [download page](#). This page contains a list of mirror sites from which you can download the setup file.

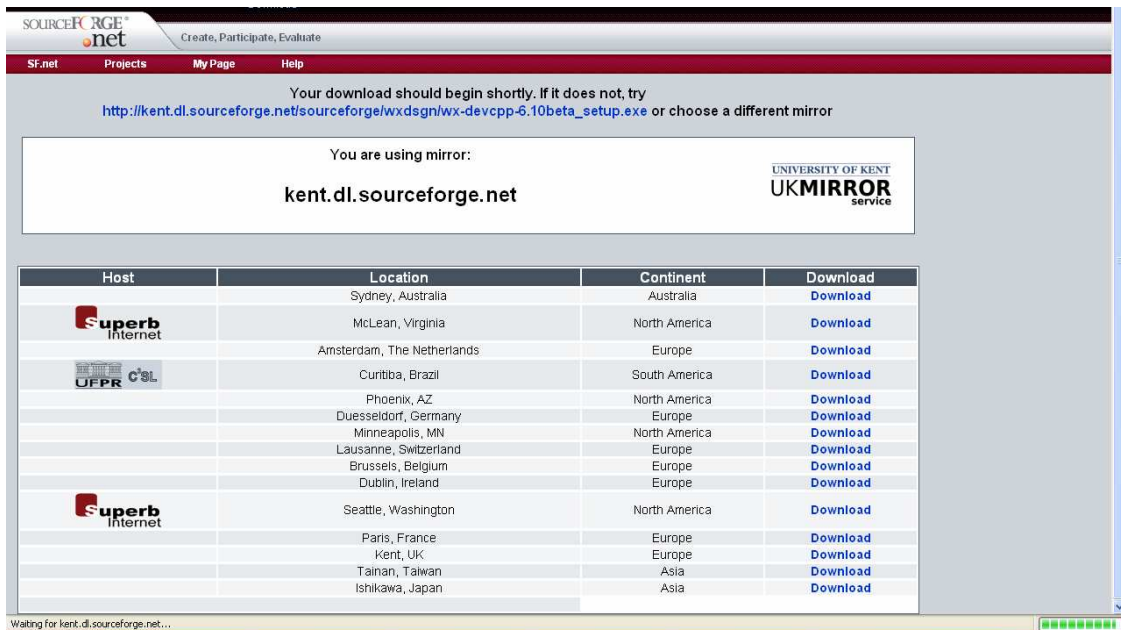


Figure 1.3 – SourceForge download page for wxDev-C++

Now choose a mirror site that is closest to your home location. For me this is Kent, U.K. To the right of the mirror site name, is a link in blue labelled “Download”.

Click on this link to access the download page from the mirror site.

The page will reload using that mirror and the download will start immediately.

NOTE: In Internet Explorer Windows XP Service pack 2, the download may be blocked. In which case it is necessary to click on the header and select ‘Allow download from this site’.

The next thing you should see is the download dialog box. This will differ from one web browser to the next, but all should contain the same basic options to either download the file or run it. If you choose the [Run] option, the setup file will download and run automatically once downloading is complete. If you choose [Save] the setup file will be saved onto your computer for you to run when you wish.

Click on either of the [Run] or [Save] buttons.

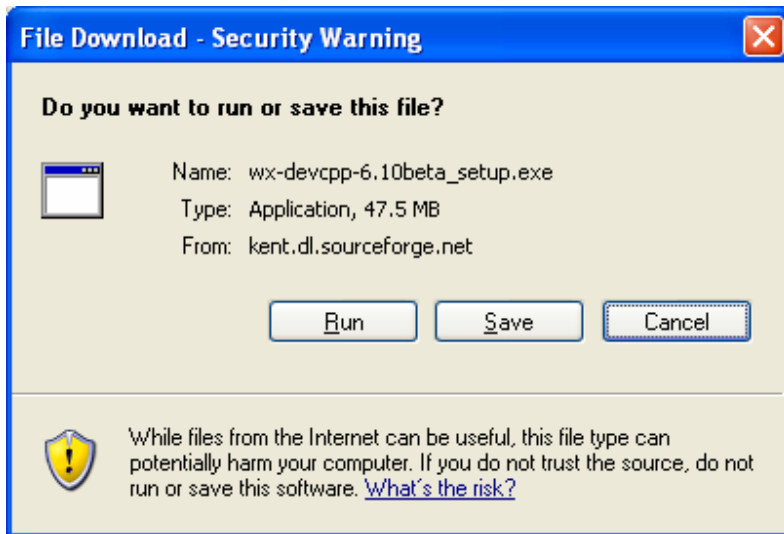


Figure 1.4 – Internet Explorer’s file download option box

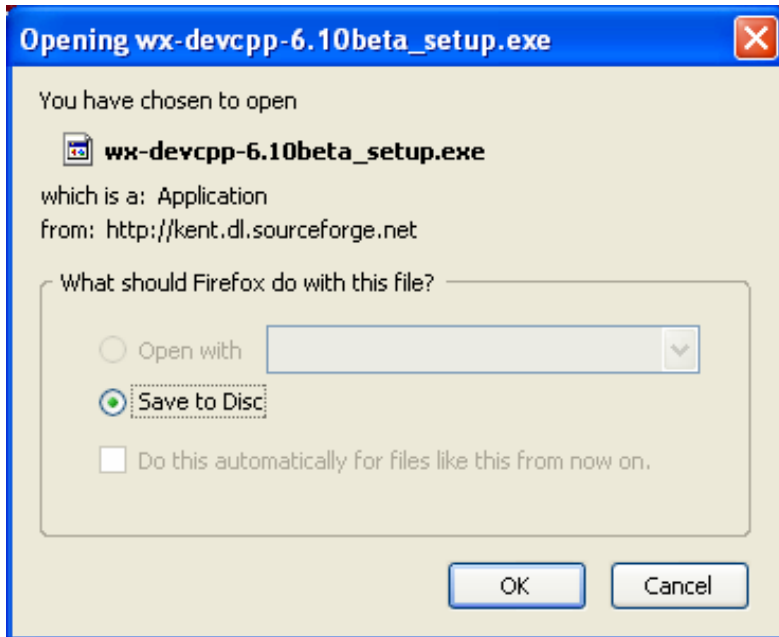


Figure 1.5 – Firefox’s file download option box

In my case I chose [Save] since I prefer to keep the setup files on hand should I need to uninstall/reinstall or install on a different computer. So click on either the [Run] or [Save] to continue.

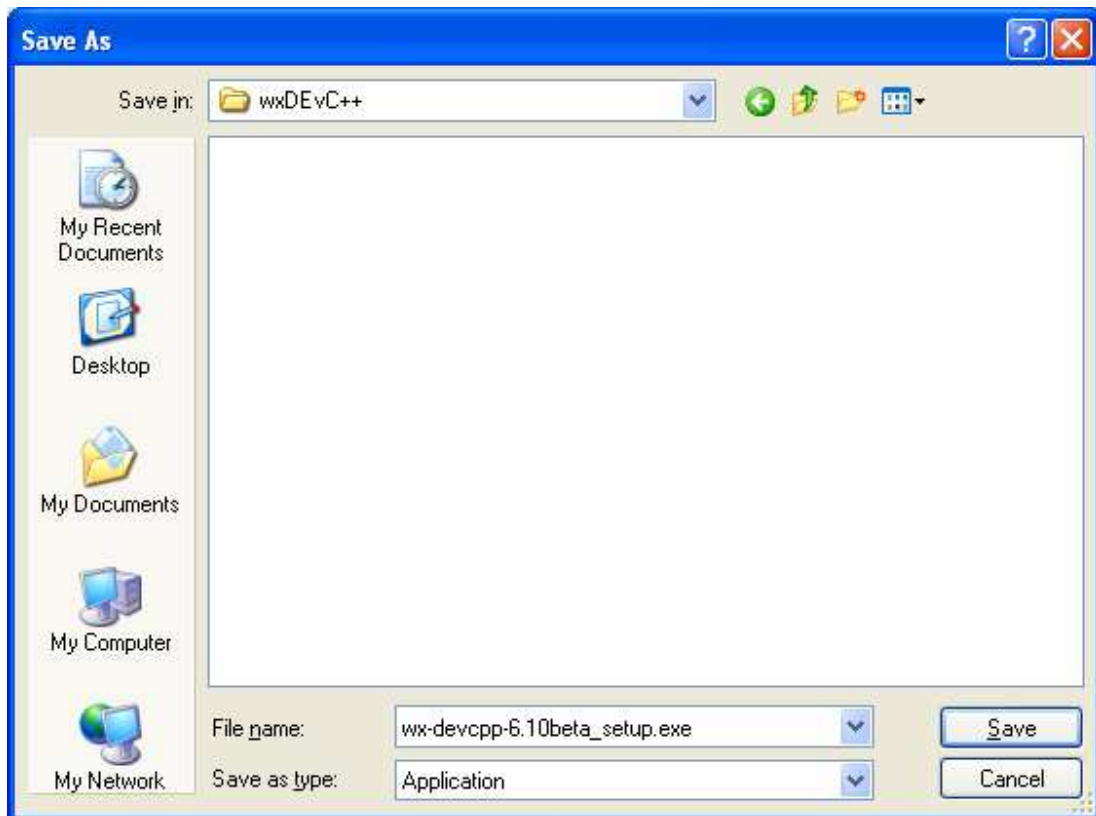


Figure 1.6 – Choosing a location to save the setup files in.

Once you have chosen to [Run] or [Save] the file, the download should begin. Depending on your Internet connection this is either time for a quick cup of tea or a quick stretch. (As the following dialog shows this download was approximately 47.5Mb in November 2006)

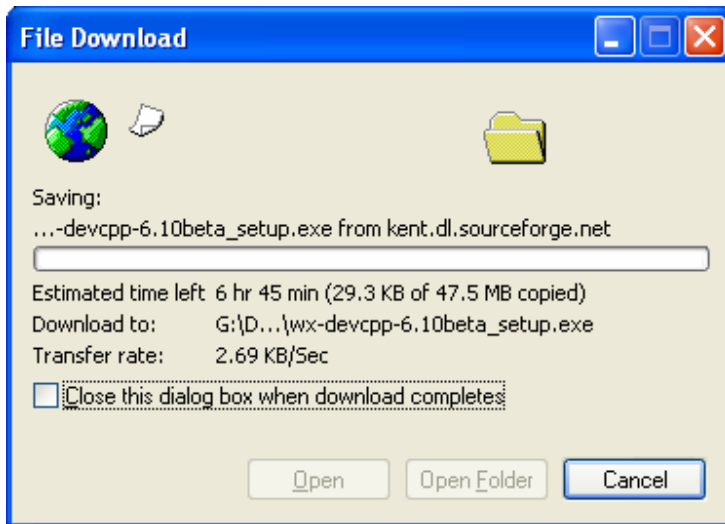


Figure 1.7 – The setup file downloading

Installing wxDev-C++

Once the file has downloaded, if you chose to run the file the installation program will start automatically, if not you need to browse to where you saved the file and:

Either double left click on the file name to run it.
Or right click and choose 'Open'.

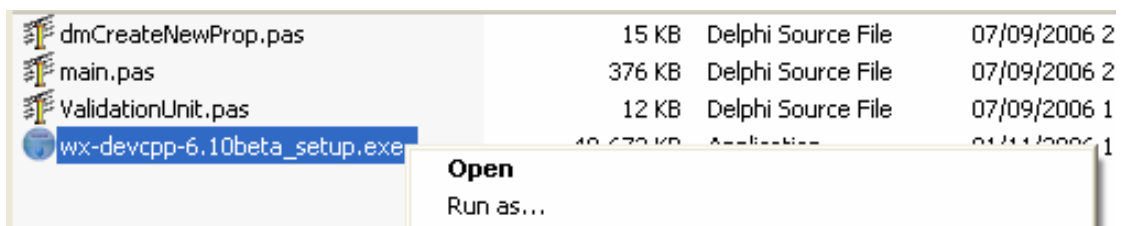


Figure 1.8 – Running the setup program

The next thing you should see is a warning box telling you not to install this program over an existing installation. **This is an important warning** as many people have found. Failure to heed these instructions can result in a broken installation which looks okay but gives you endless headaches (not unlike cheap cider). See the section Updating wxDev-C++ later in this chapter for further details.

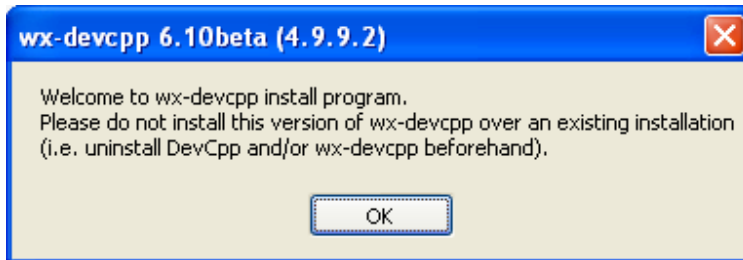


Figure 1.9 – The initial warning message from wxDev-C++ setup

The next dialog displays the language selections used during the install. Personally I stick with the default English since I have trouble understanding anything else.

Select your language and then click [OK].



Figure 1.10 – Choose a language option dialog

The next option marks a change from previous versions of wxDev-C++. Since this release offers support for more than one type of compiler, it offers you the choice of which compilers you wish to use.

Check the boxes next to the compiler you wish to use. Then click [Next>]

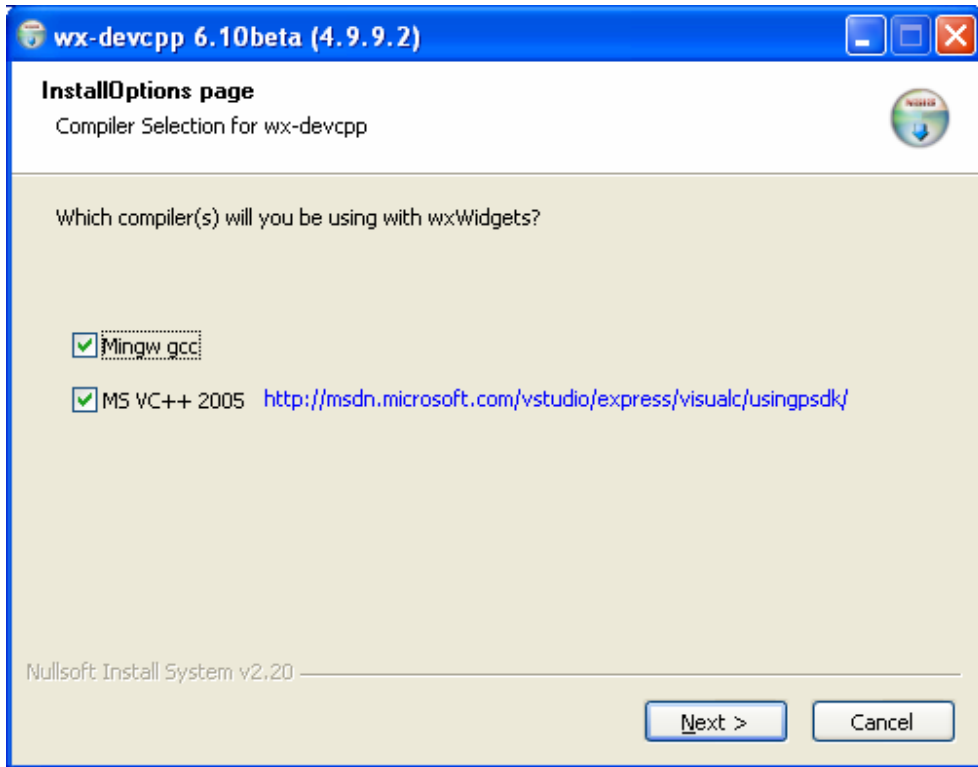


Figure 1.11 – Choosing which compilers you wish to use

Although wxDev-C++ is free, to use it you must agree to the provisions of the license agreement. The license used is the GNU GPL (General Public Licence) Version 2. It is up to you to either read through all of it, or skip it.

Click on [I Agree] to continue any further.

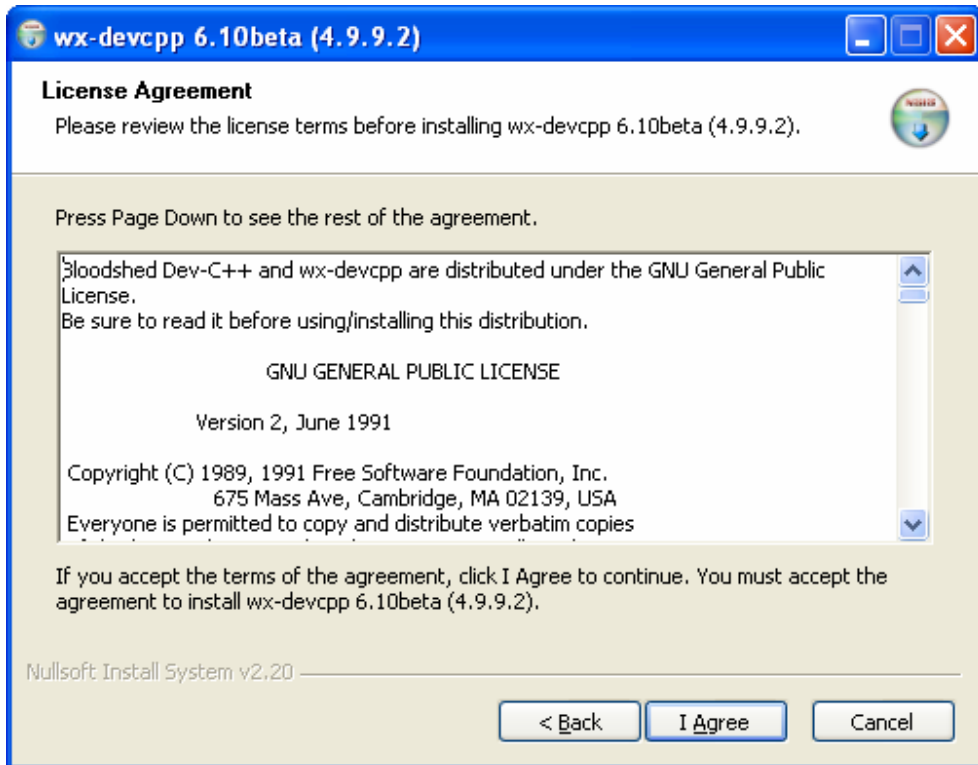


Figure 1.12 – The license agreement dialog

The next dialog offers you the chance to choose which components you wish to install. Personally I keep to the defaults, but it is worth scrolling through the list of components to get some idea of what is included in the distribution. The top combo box labelled ‘Select the type of install’ gives you an option of full, minimal and custom install. Use the minimal install if space is at a premium on your computer.

You will notice that the first two choices are grey this is because they are required to actually install something of use. If you use another IDE (not that you would) and you are only trying wxDev-C++ then you may wish to uncheck the option to associate files with wxDev-C++. Likewise if you didn’t make the change on last screen this is your last chance to choose which compilers you will support and load the libraries for those.

Make your choices and then click on [Next >] to continue.

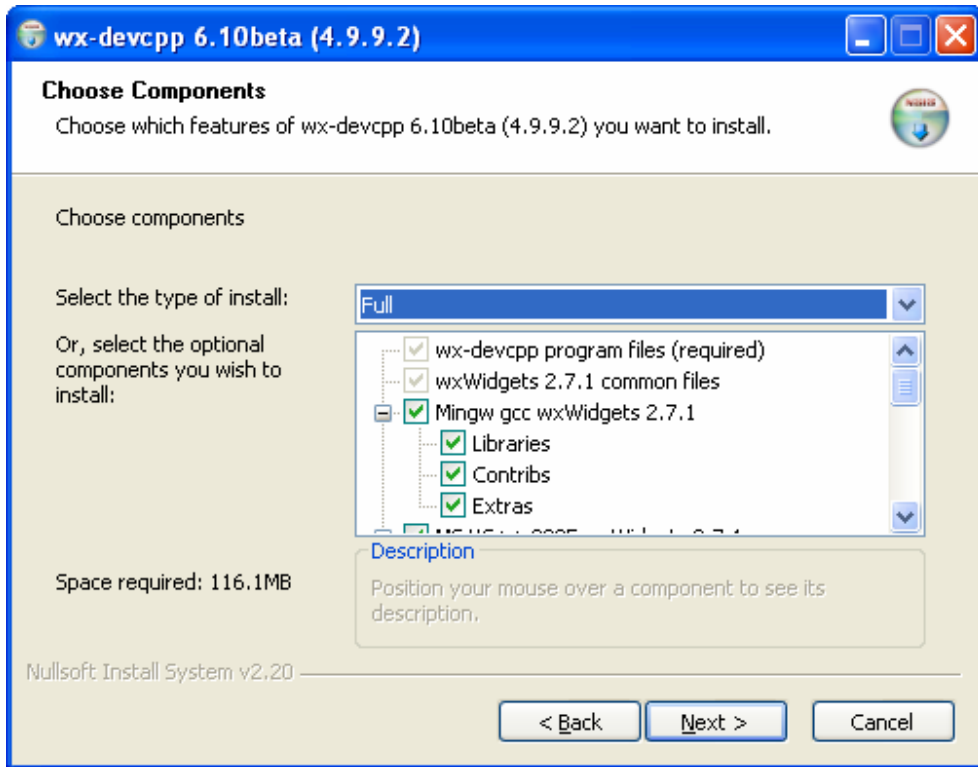


Figure 1.13 – The component choice dialog

The Start Menu option dialog gives you the option to choose where wxDev-C++ appears on your start menu. Again, this is a personal choice. I like to group similar types of programs together, so I change this option to 'Programming\wxDevCpp'.

Make your choice and then click [Next >] to continue.

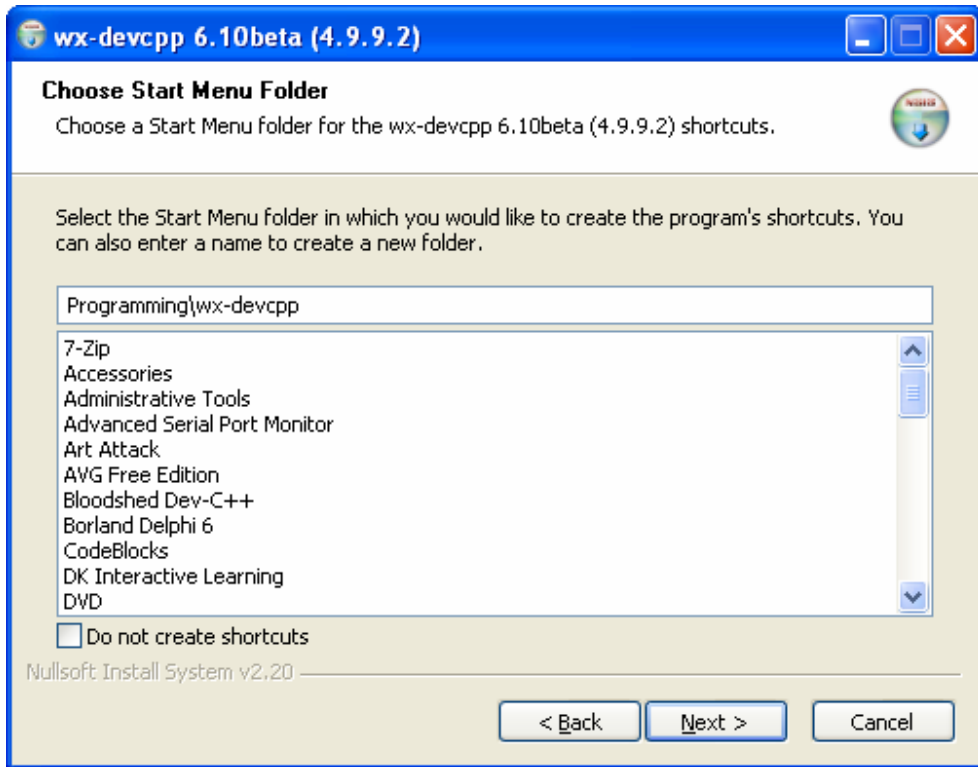


Figure 1.14 – Start menu location option dialog

The Install Location dialog provides the option of setting the wxDev-C++ installation location. This is one default I usually stick with. Previously this used to be 'C:\Dev-Cpp' due to the fact that DevCpp could not handle spaces in the file path when compiling programs. However due to the hard work by wxDev-C++'s developers this is no longer the case. Hence the default is now 'C:\Program Files\Dev-Cpp'. Just one of the many improvements in this release.

Breathe in once and click on [Install].

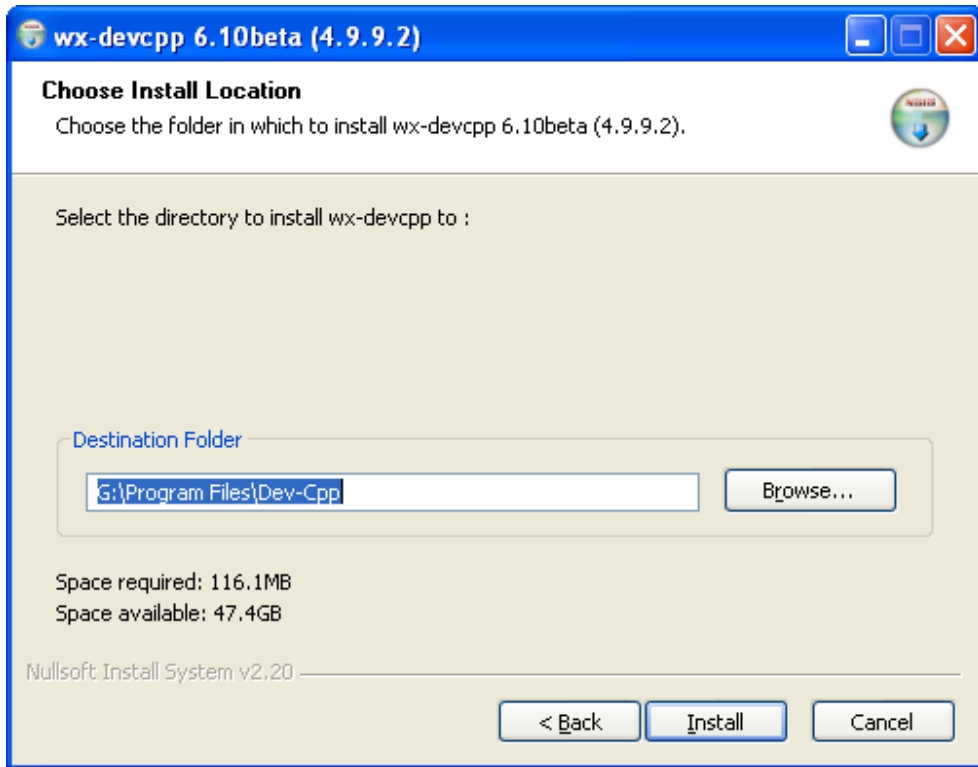


Figure 1.15 – Choose an install location dialog

While the next dialog fills with the names of all the files being installed you will have time for another brief break.

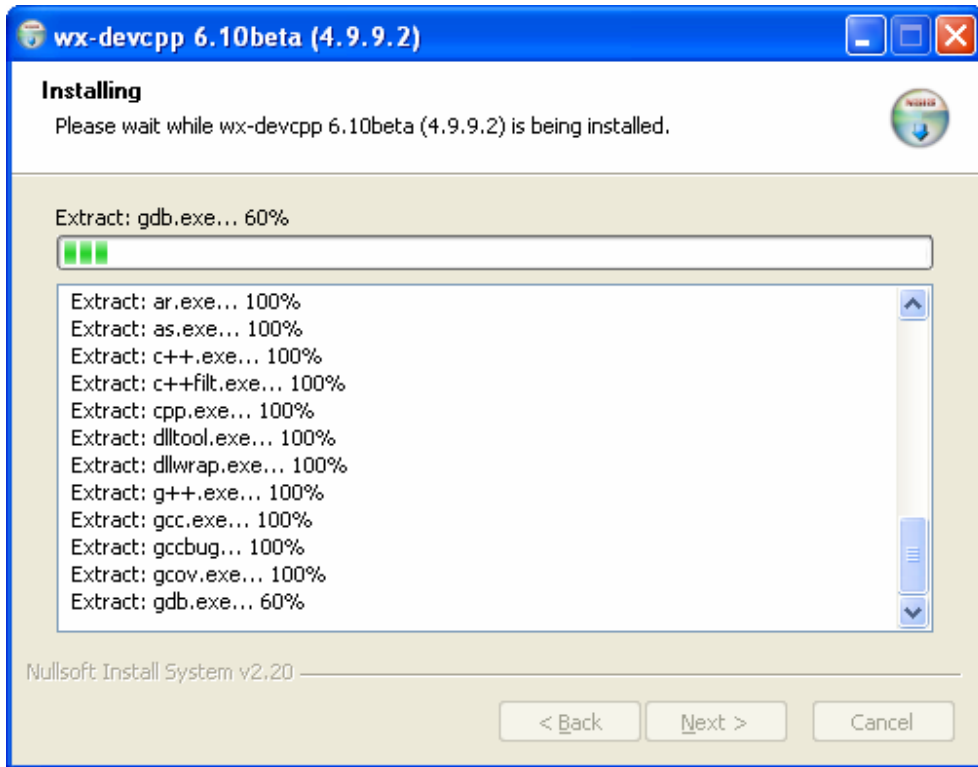


Figure 1.16 – File Installation dialog

Halfway through the file installation, the following dialog will pop up. If you want an entry for wxDev-C++ on the start menu for all users on your computer then select [Yes], otherwise select [No]. I select [No] since the other users of my computer don't want me messing up their environment for them.

Click on either [Yes] or [No] to continue.

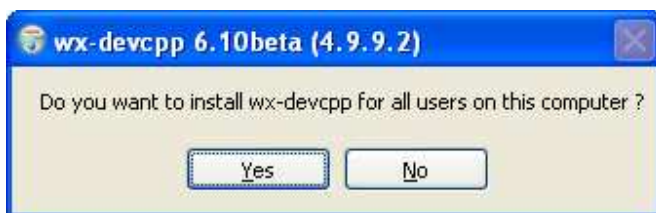


Figure 1.17 – Install for all users dialog

More files will scroll past. Soon wxDev-C++ will finish installing all the files it needs.

Now click [Next] to continue.

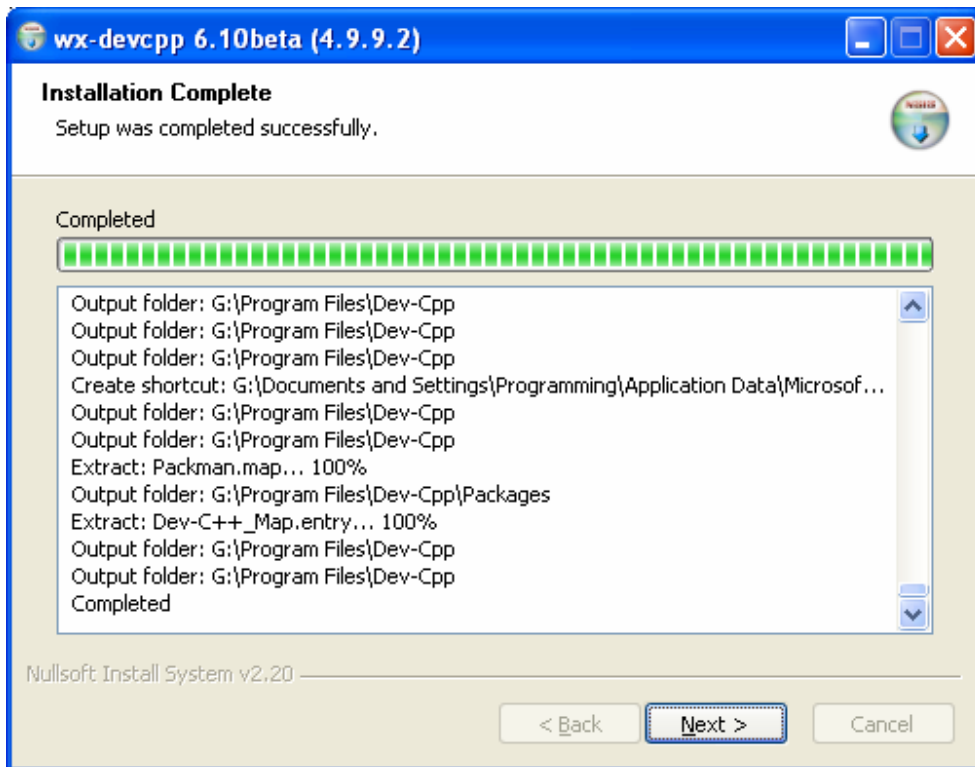


Figure 1.18 – File Installation dialog upon completion

This will lead you to the Completing Setup dialog. Untick the check box labelled ‘Run wx-Dev-C++’ if you don’t want wxDev-C++ to run when you exit the wizard. Equally untick ‘Read Sof.T’s wxDev-C++ Programming Manual’ if you don’t want to read this book.

Preferably leave ‘Run wx-Dev-C++’ checked and continue following these instructions. If you don’t, the next time you run wxDev-C++ you will need to complete the next steps.

Click [Close] to exit and cheer loudly.

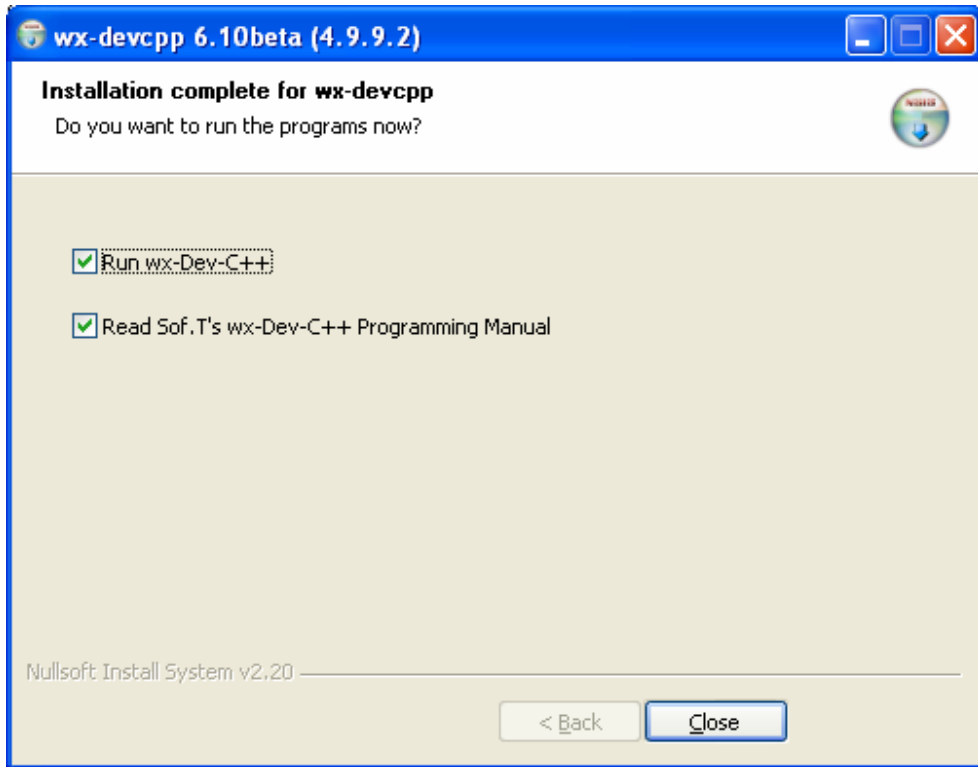


Figure 1.19 – The completion of the setup dialog

When wxDev-C++ is started up for the first time you will currently be greeted with the beta warning dialog. This will be absent on later full release versions. Read it or not as you wish.

Click [OK] to continue.

It is worth remembering that bugs can be posted as you find them. This feature enables Open Source software to continue improving and gives you something better to do than just cursing the programmers when the application crashes. The other point to note is that of updates, which is covered in detail later in this chapter, in the section 'Updating wxDev-C++'.

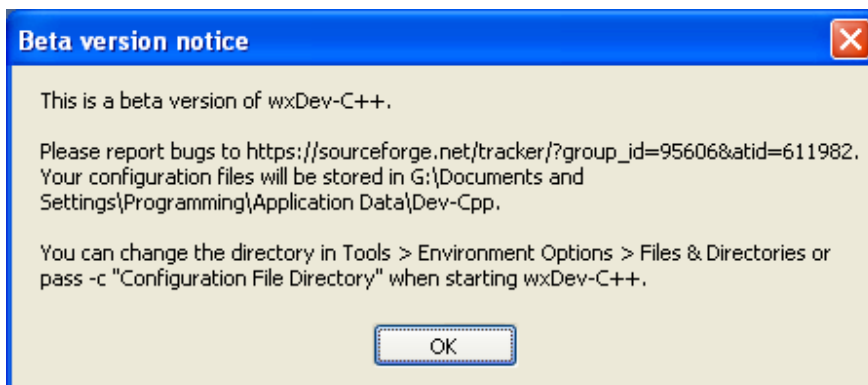


Figure 1.20 – The beta dialog

You are now presented with various options to customize your version of wxDev-C++. This is the same introduction as the standard version of DevC++. Here you can choose your preferred language. As mentioned earlier I stick with English. You can choose between 3 different icon themes, (I prefer New Look) and choose whether or not to support XP Themes.

Make your customisation choices and click [Next] to continue.

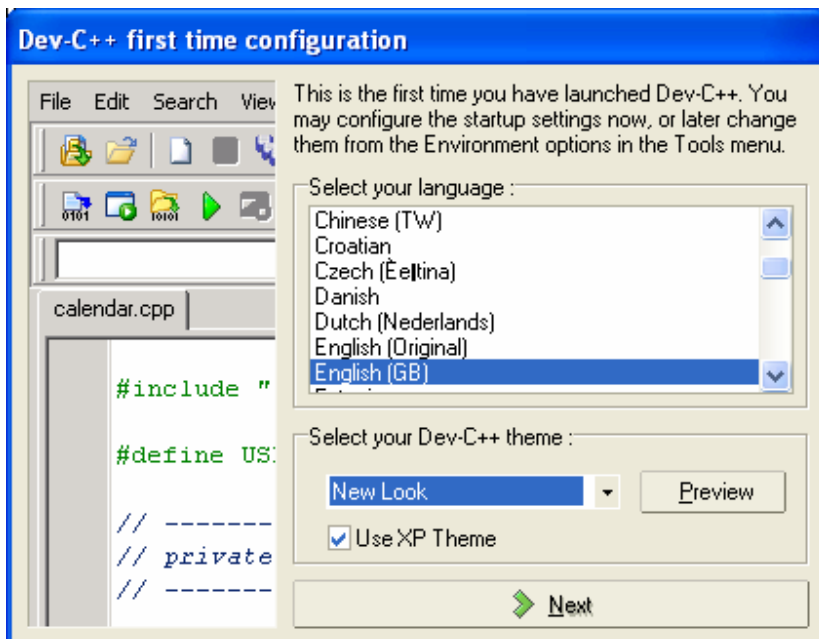


Figure 1.21 – Configuration Dialog

Next you have the option to enable code completion. Choose ‘Yes, I want to use this feature’. The visual designer in wxDev-C++ relies on the code completion feature to automatically create events for you. While code completion can be annoying when it seems to get in the way, it is also a great source of information, and helps avoid typo errors.

Click on [Next] to continue.

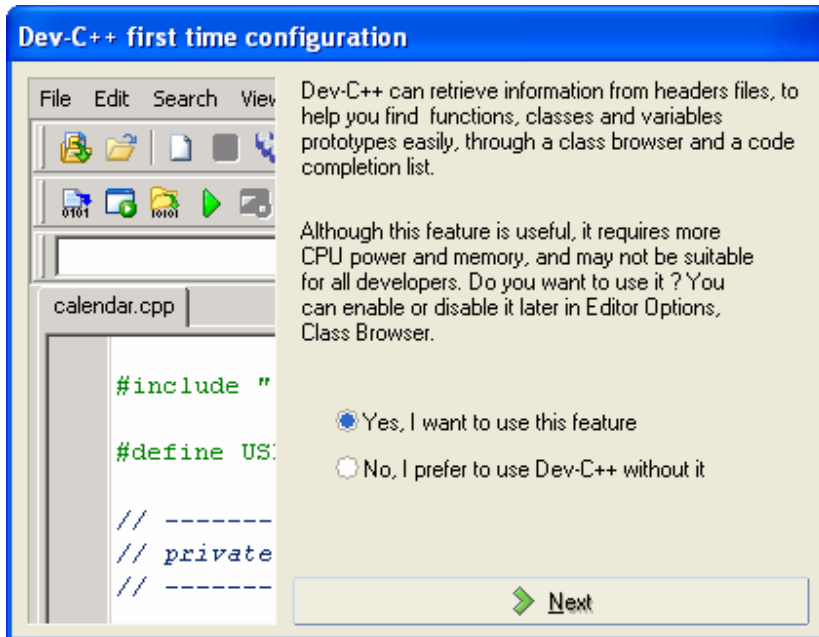


Figure 1.21 – Enable code completion dialog

The second part of the code completion feature asks if you want to create a cache. Basically this scans through all the .h header files in your include directory and builds a list of functions, etc. Later when you are programming code completion uses this cache to prompt you or to complete the code.

Select the option ‘Yes, create the cache now’.

Click [Next] to continue.

At this point, unless you have a very fast computer, go and put the kettle on and brew a coffee.

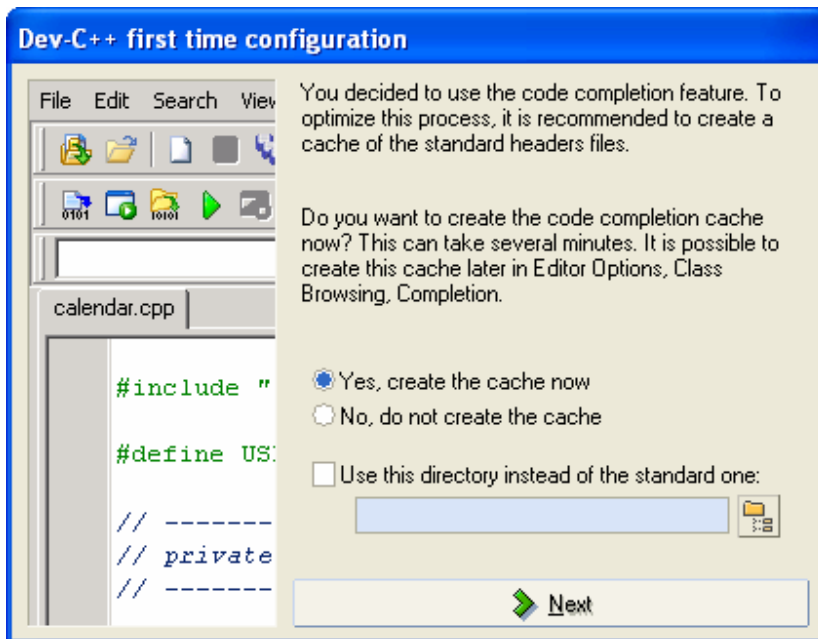


Figure 1.22 – Code completion cache creation dialog (try saying that fast)

Drink your coffee and continue waiting. It does end eventually I promise.

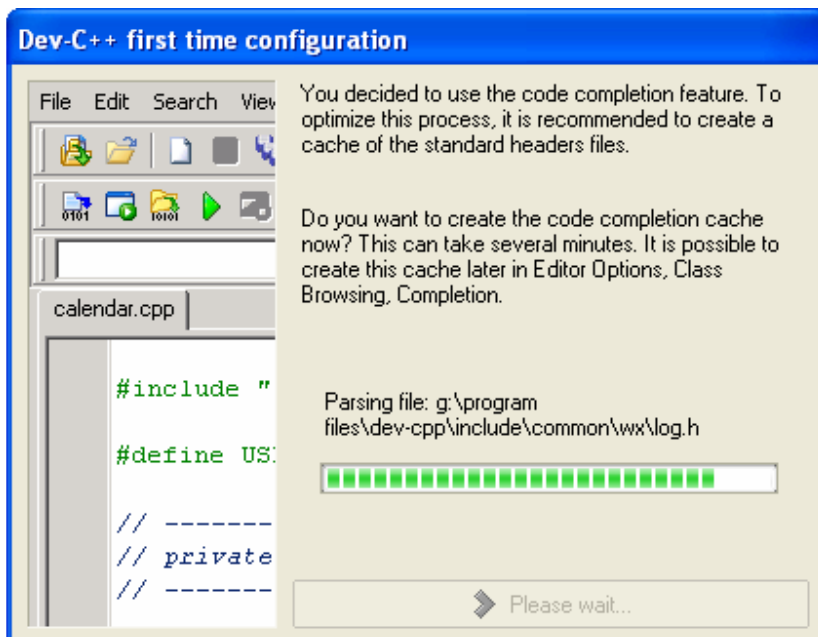


Figure 1.23 – Yep still waiting, nearly finished that coffee though

Hooray we have made it to the final dialog. Read or not as you wish and:

Click [OK] to complete the installation process.

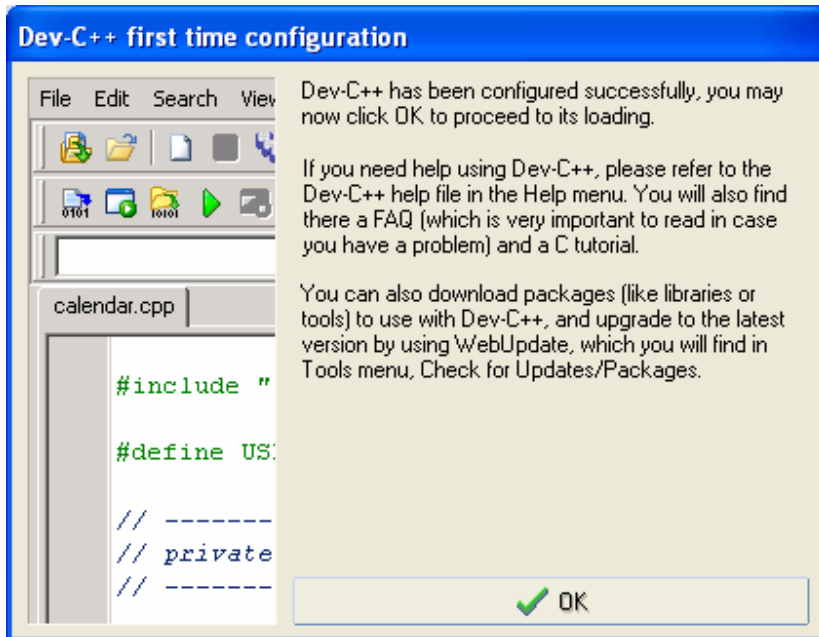


Figure 1.24 – Phew the final dialog

After a short pause the IDE will appear, followed by the tip of the day window.

Updating wxDev-C++

Updating wxDev-C++ is a fairly simple procedure as long as you remember that dialog box that appeared during the install saying ‘Please do not install this version of wxDev-C++ over an existing installation’. For old hands at windows this will be a simple procedure, but just in case you are not sure, here is how to proceed.

As always click on the [Start] button on the toolbar.

From the pop-up menu select the ‘Settings’ menu

Click on ‘Control Panel’

(or on WindowsXP click direct on the option ‘Control Panel’).

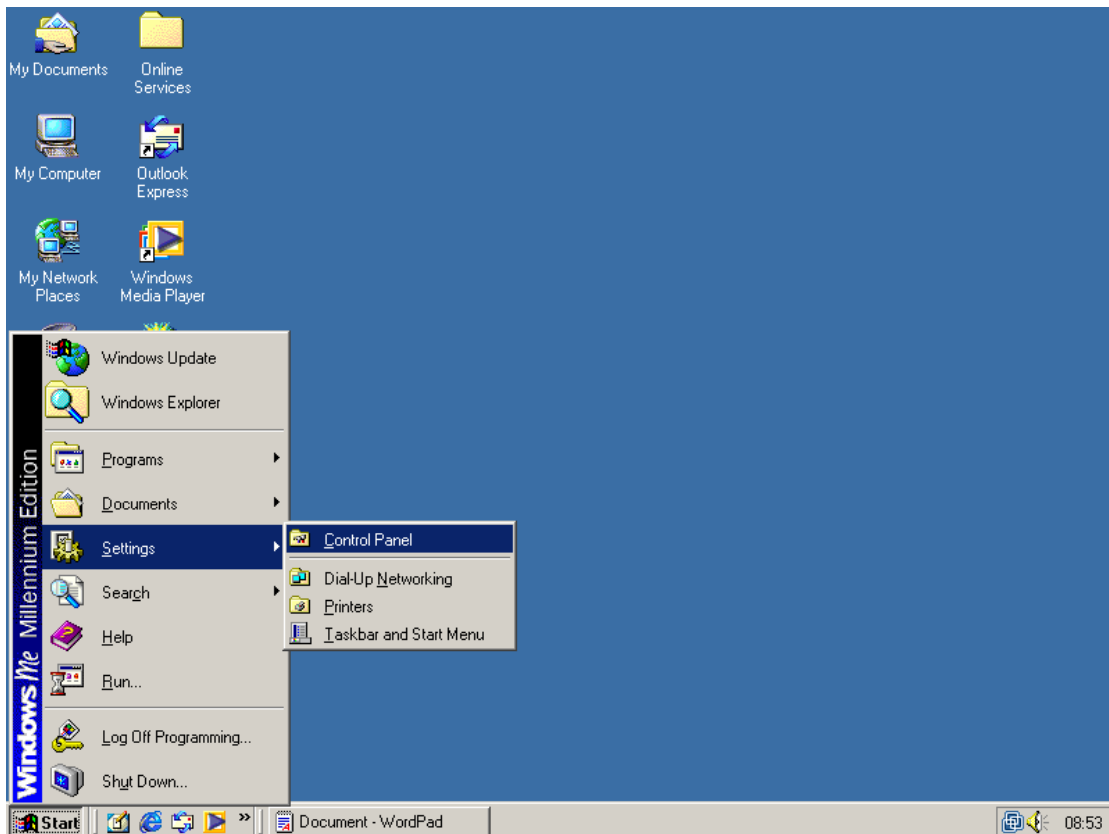


Figure 1.25 – Getting to the Control Panel (Windows 9x)

The Control Panel will pop up with various options.

You need to select Add/Remove Programs.

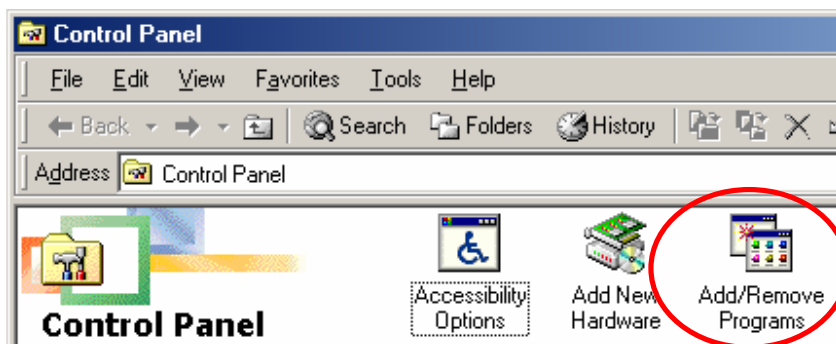


Figure 1.26 – Selecting Add/Remove Programs

The Add/Remove dialog will appear. Depending on how many programmes you have installed on your computer, it may take a few seconds to fill the dialog. When it appears scroll down the list until you get to wx-devcpp. Select this option; once it is highlighted the Add/Remove button will become activated.

Click the [Add/Remove] button to continue.

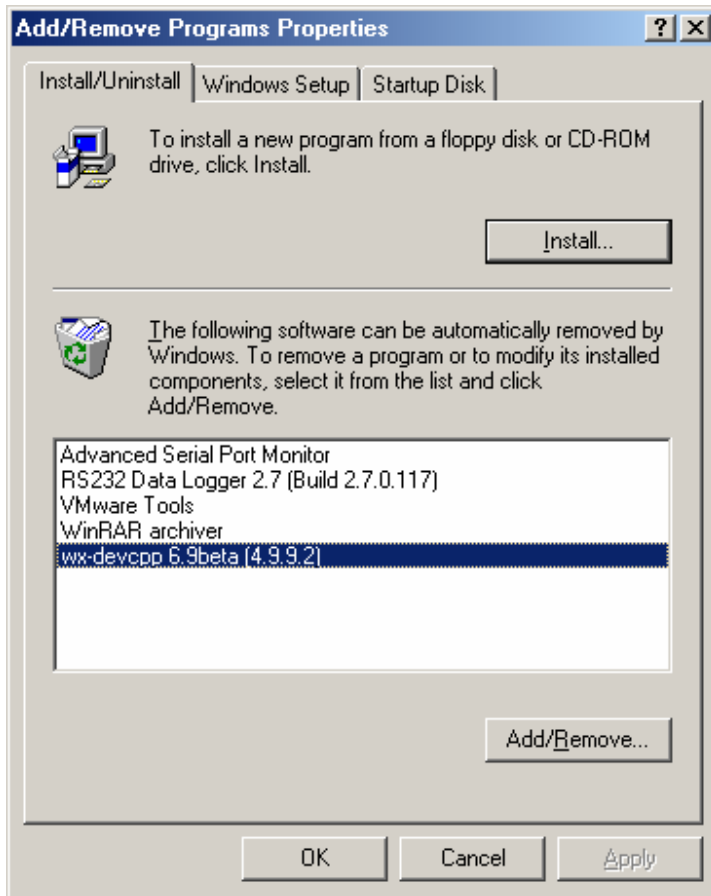


Figure 1.27 – The Add/Remove Dialog

The next dialog shows the location of wxDev-C++ and you need to click 'Uninstall' to continue.

Click the [Uninstall] button to continue.

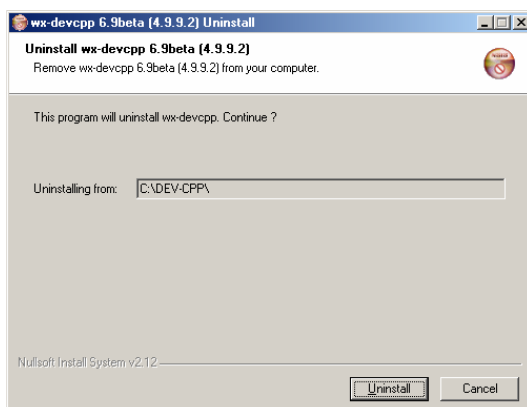


Figure 1.28 – Uninstall dialog

A list of files is displayed as they are uninstalled.

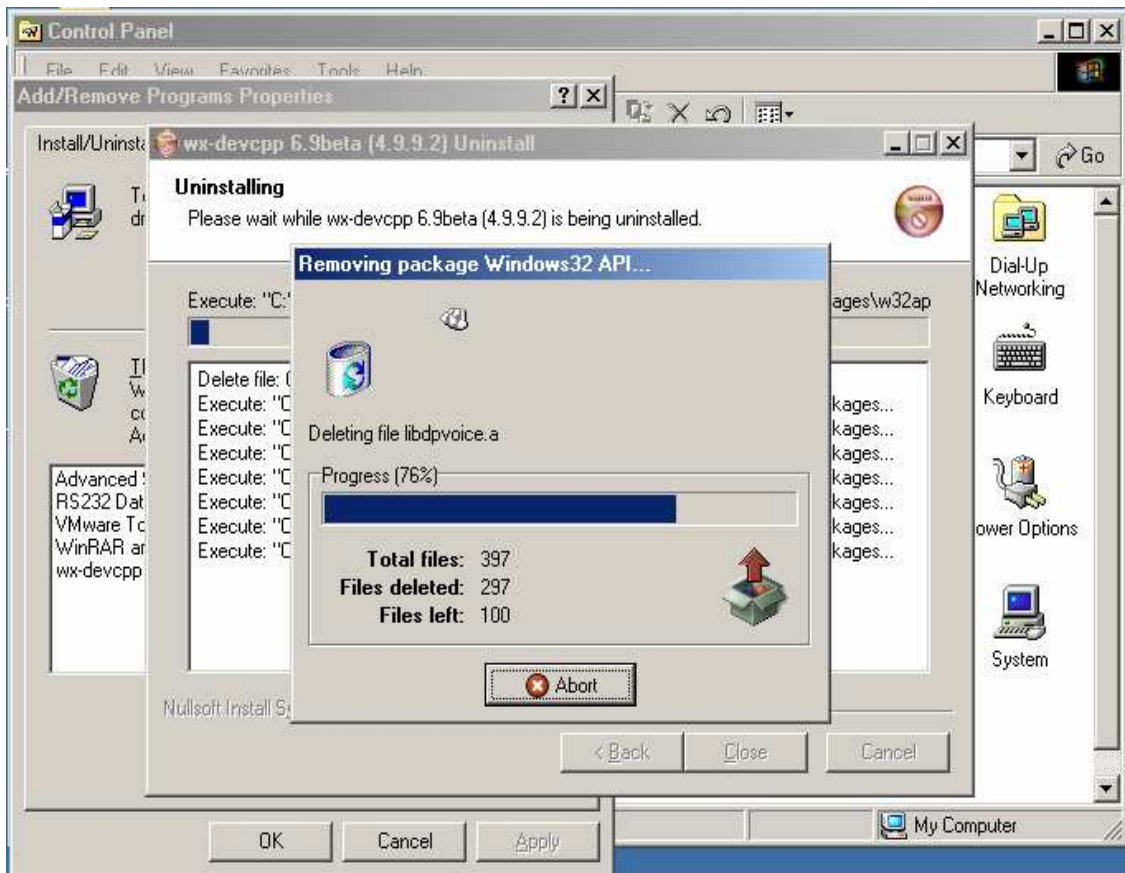


Figure 1.29 – wxDev-C++ being uninstalled.

Finally the following dialog will pop up. If you have spent a lot of time configuring the IDE to your own preferences, you may wish to keep the configuration files.

Click the [No] button to keep your configuration files

or

Click the [Yes] button to delete your configuration files and revert to the default.

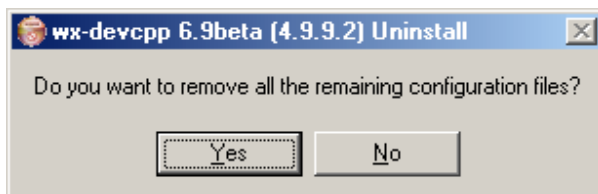


Figure 1.30 – Remove configuration files dialog

At last, the final dialog. Since I personally keep my projects in C:\DEV-CPP, I leave this directory alone. It is safe to leave this directory to install your new version into. Or delete it. As ever, the decision is yours to make.



Figure 1.31 – The final dialog

To install the latest release from the site on Sourceforge follow the instructions in the prior section ‘Getting wxDev-C++’.

Advanced Users

For advanced users there is the option to try the latest cutting edge versions of wxDev-C++ alpha builds. These builds show features that may make it into future versions of wxDev-C++. There are two sites to try these from, [Tony Reina’s](#) site and [Joel Low’s](#) site. Both are accessible from the wxDev-C++ home page under ‘Alpha builds’ on the side navigation bar.



Figure 1.32 – Link to alpha build of the wxDev-C++ IDE

Below is the title page for Tony’s site. Tony generally has various versions of wxDev-C++ available.



Figure 1.33 – Tony’s wxDev-C++ page

Click on the link to wx-devcpp Testers (alpha versions). And it will take you to the following page. You will want to grab the devcpp.palette as this adds the latest controls to the widget palette in the IDE. The dates alongside the files give some indication as to which are the latest. Click on the one you are interested in. Save it the location where you installed wxDev-C++ (This is ‘C:\Program Files\Dev-Cpp’ if you accepted the default location).

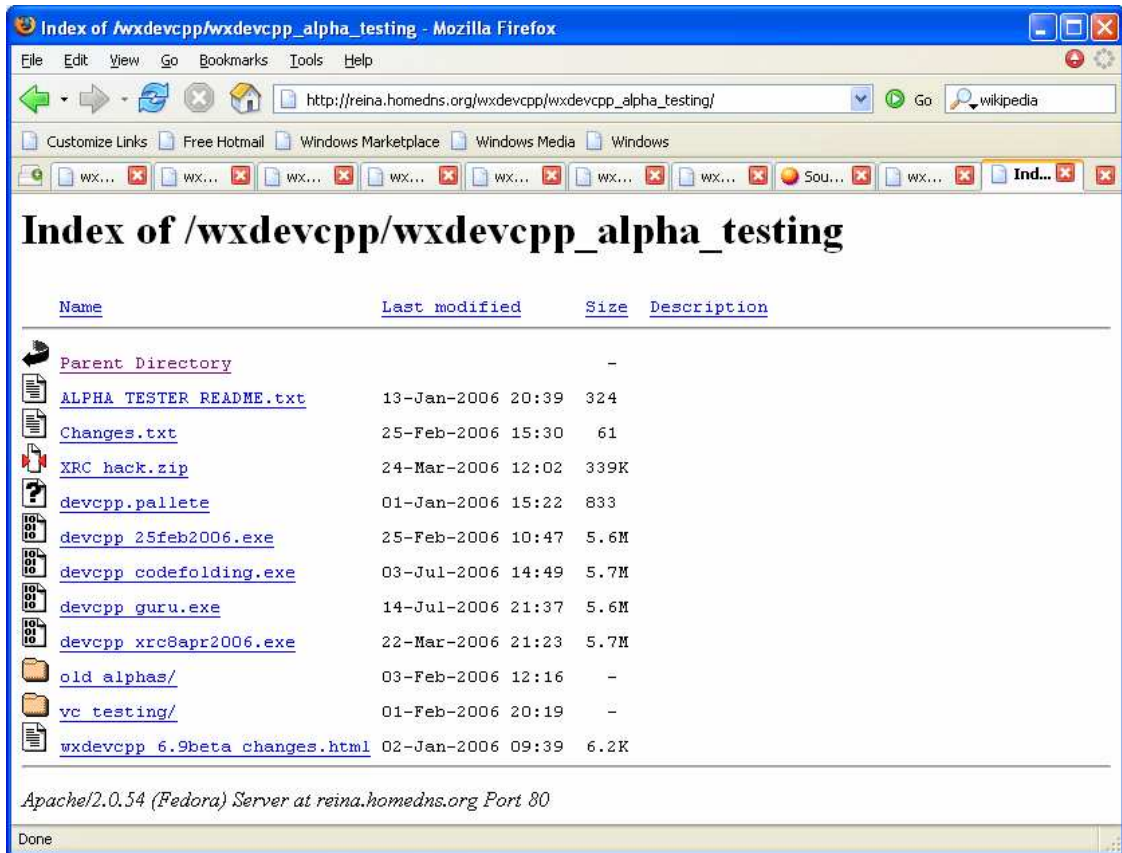


Figure 1.34 – Tony’s list of wxDev-C++ versions

When it has finished downloading browse to `C:\Dev-Cpp` and rename the file `devcpp.exe` to something else like `devcpp.exe.backup`. This makes it available to change back if the new version is too unstable. Now find the file you downloaded and rename this to `devcpp.exe`. You can now start wxDev-C++ as normal.

Joel’s page on the other hand has links to alpha versions in binary only or installer packages. He also lists links to various prebuilt wxWidgets libraries. These are also available via web update. See the next section for more details.

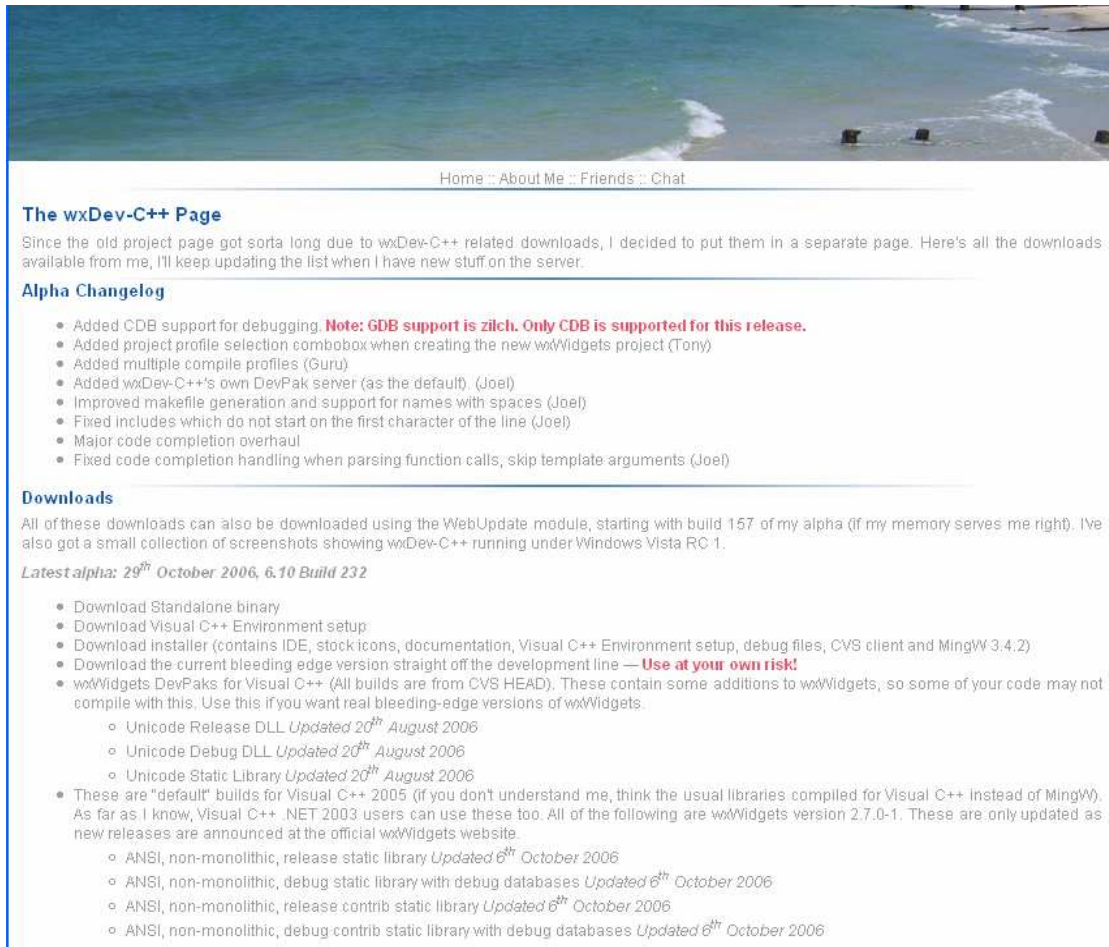


Figure 1.35 – Joel's list of wxDev-C++ versions and packages.

The latest CVS version should be located at

<http://home.wanadoo.nl/m.nealon/devcppcv.exe>. Which gives you another option to try.

Adding Extra Packages

Somewhere during the development of DevC++ someone decided that it would be good if users could add to the libraries they used and update the IDE. The first version of this updating mechanism was called VUpdate. However this was eventually scrapped and DevC++ moved onto a new system called Web Update. This allows the user to download newer versions of DevC++ as they are released and to download DevPaks. DevPaks are file packages which contain many different things from help files to extra libraries. Since wxDev-C++ is based on DevC++ it uses the same Web Update feature, but as a result there are some pitfalls to watch out for as we will discuss later.

Firstly let us look at how to add extra libraries to our new installation of wxDev-C++.

If wxDev-C++ is not already running, start it up.

From the 'Tools' Menu select 'Check for Updates/Packages...'

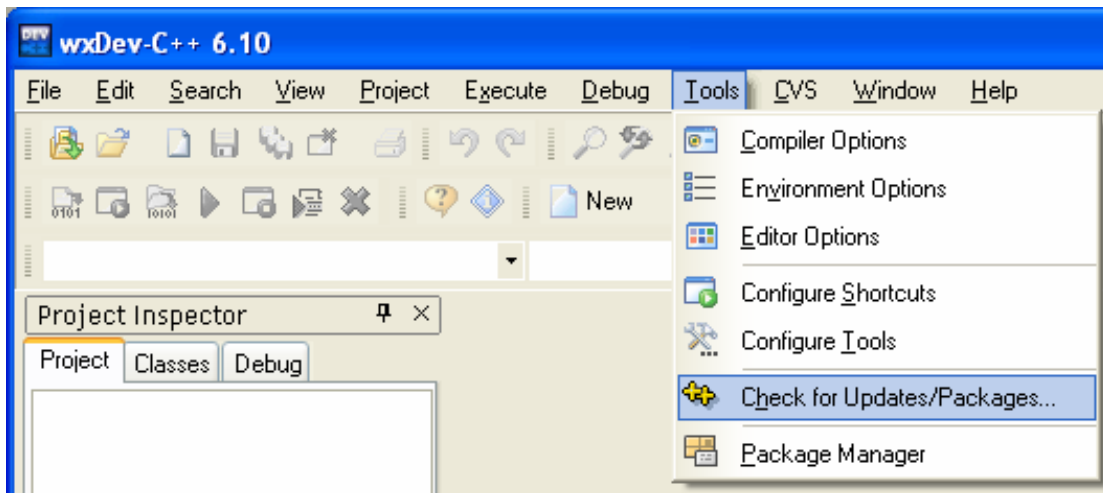


Figure 1.36 – Check for updates from the tool menu

This will activate the Web Update application. Make sure you are connected to the Internet before proceeding any further. At the top of the dialog is a drop down listbox labelled 'Select DevPak server'.

Click on the arrow to reveal a list of servers (Currently only there are three, only one is dedicated to wxDev-C++).

Select 'Dev-C++ primary devpak server'.

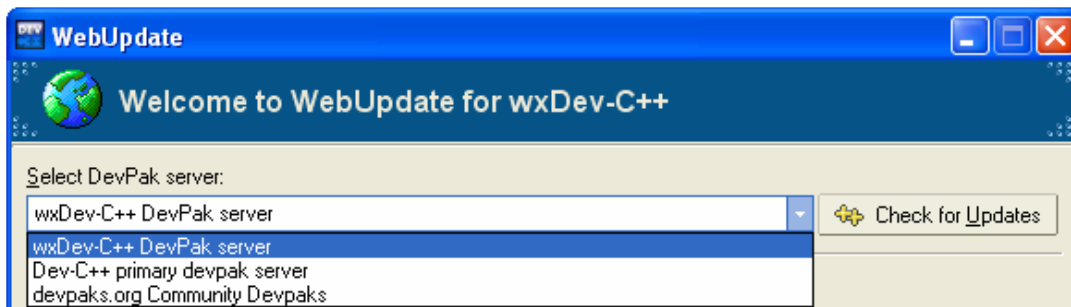


Figure 1.37 – Selecting a DevPak server

Once you have selected the server:

Click the [Check for updates] button at the bottom of the dialog.

After a short pause the main part of the dialog should be filled with a list of updates you can download.

From left to right the list tells you the name of the update, the version number of the update, and, if you already have this file installed, the version number of the installed file. This saves you from downloading and installing out of date versions. Next follows the

size of the file which can be handy on limited bandwidth connections. Finally the file creation date gives you some reference point on whether the file is up to date or not.

Click on the check box next to the file name to select files for downloading. To activate the download, click on the [Download Selected] button.

Note the warning below....At this point all files with a green check next to them will be downloaded.

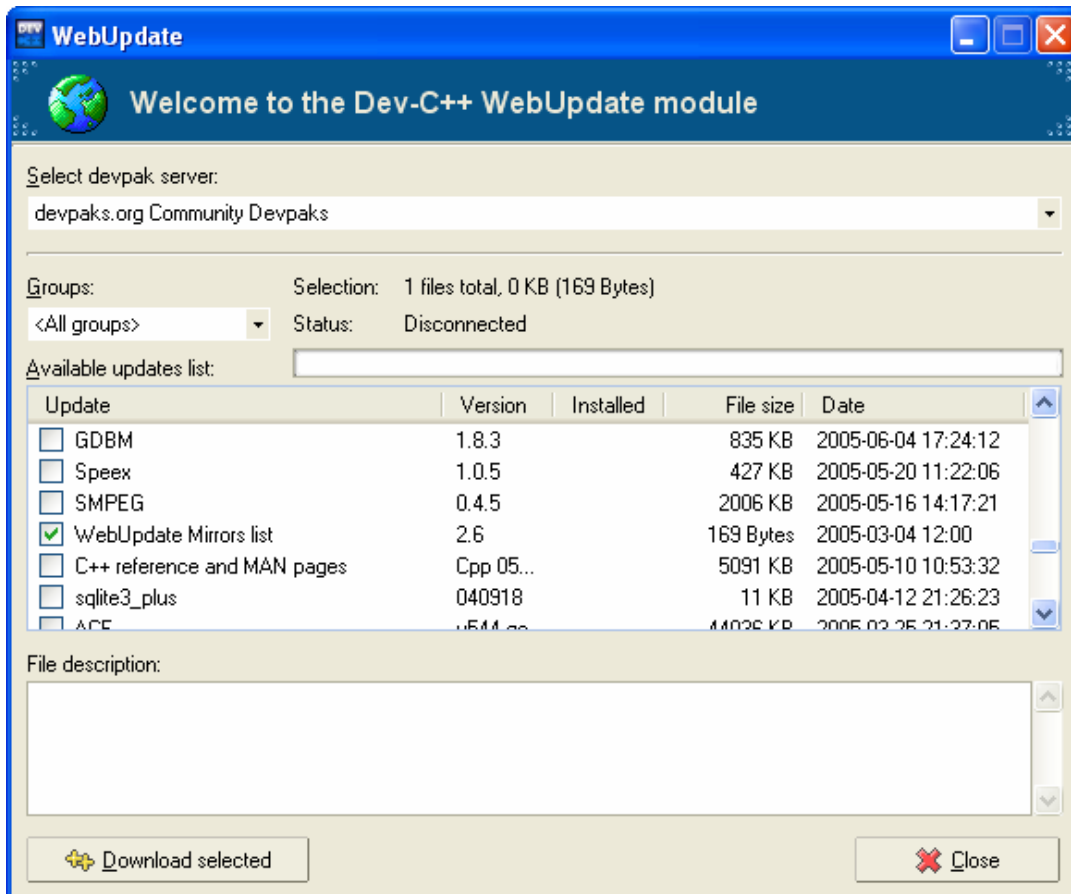


Figure 1.38 – Selecting libraries to download.

WARNING: Remember this system is used to update DevC++ as well as wxDev-C++. The following image shows a new version of DevC++ that can be downloaded. Do not download this or you will lose the visual designer part of DevC++.

Problems can also arise when downloading versions of wxDev-C++ that are marked Alpha. Alpha versions may be less stable than your current version or remove some features.

Equally do not download libraries called wxWindows, this is the old name for wxWidgets and will cause you a headache. Finally be wary when

downloading versions of wxWidgets libraries, if they are compiled with different or bad options they can break a healthy installation. It is safest to download from the wxDev-C++ server.

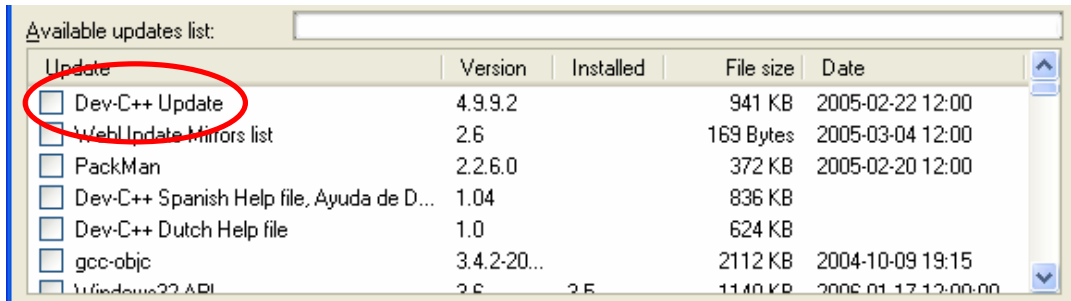


Figure 1.39 – Careful not to download updates of DevC++

When your files have finished downloading either they will be installed quietly as in the case of WebUpdate Mirrors, or the following dialog will popup.

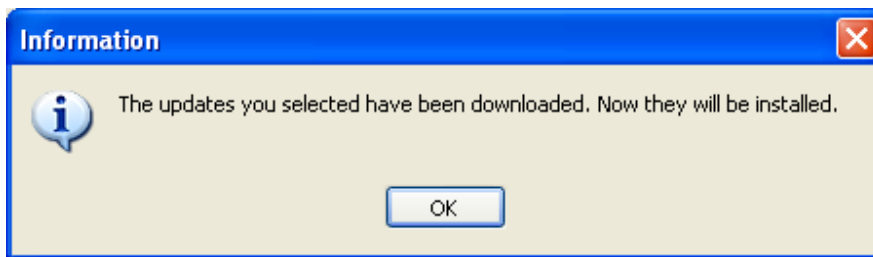


Figure 1.40 – Installing updates dialog

Clicking 'OK' starts up the installation part of another application called PackMan. No this is not a small yellow ball with a big appetite, it is the Devpak manager. Here you can choose either [Install >] or [Cancel]. Most frequently you will want to install.

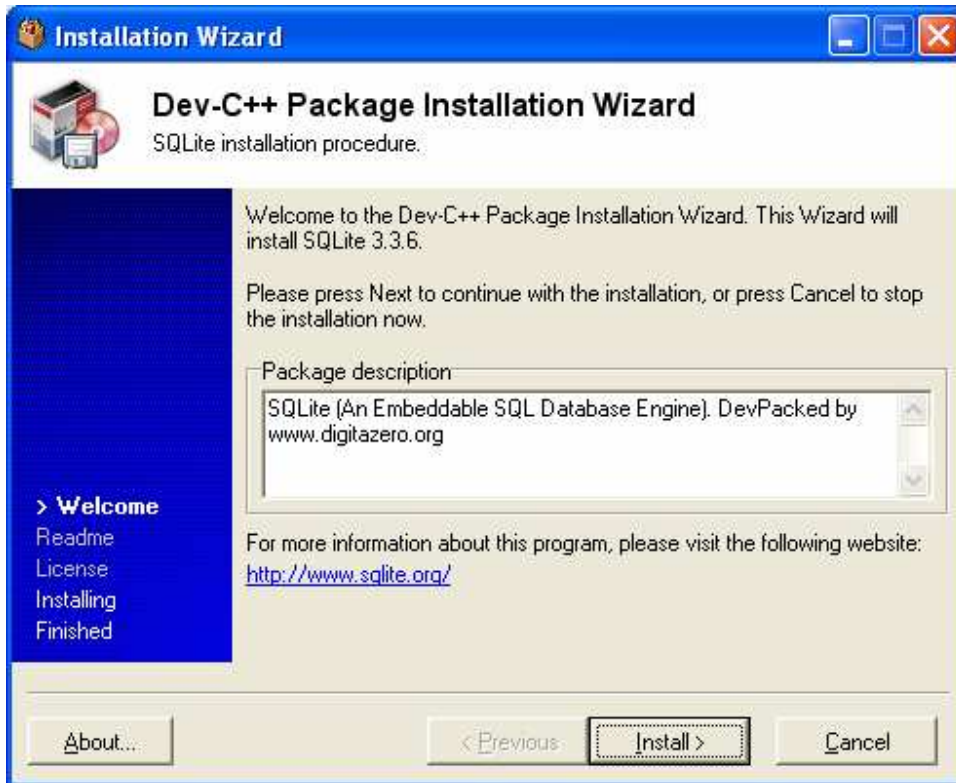


Figure 1.41 – Installing the new package.

You have now installed your new package. You will find that it may have added new templates to your New Project dialog, new help files to your system or even new libraries to play (or work) with.

Package Maintenance

But what if you wish to remove a package you downloaded? Or to check what packages you have available. This is all possible from within PackMan. To do this

Select 'Package manager' from the 'Tools' menu.



Figure 1.42 – Checking your packages

Once PackMan starts it lists all of the available packages. As you click on each one, the panel on the left alters to tell you the package name, the version number, a brief description and a reference website. If you click on the tab next to 'General' on the left called 'Files' it will list all the files contained in this package.

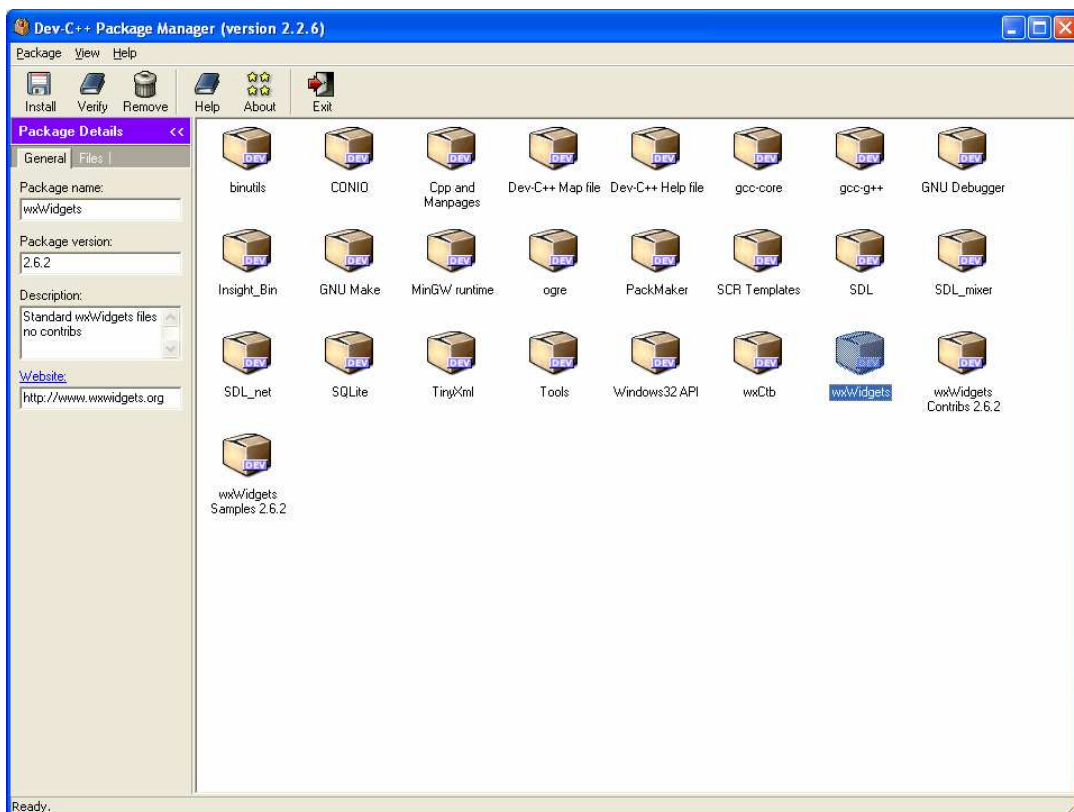


Figure 1.43 – Controlling your packages

It is also possible to install and remove packages from here.

The above procedure is not the only way to get and install new devpaks. It is possible to download devpaks from various websites. To get some idea of the variety available, type 'devpak' into google and run a search. One of the sites returned will be Devpaks.org. You may remember seeing this name on the drop down list on Web Update. Devpaks.org is one of the largest sites for locating devpaks.



Figure 1.44 – The home page for Devpaks.org

The packages are listed under various categories. It is quite possible to download devpaks from here. Once downloaded you can run the files or browse to them and double click them. If wxDev-C++ has been properly installed this will automatically start the installation wizard you saw earlier.

Other smaller sites exist for wxDev-C++ related devpaks. Such as the one mentioned on the forum shown below. This site can be located here <http://mirror.cdhk.de/wx/>



Figure 1.45 – Announcement of a new wxDev-C++ devpak site

Other devpaks are available such as these from NinjaNL

<http://home.wanadoo.nl/m.nealon/wxWidgets-2.6.2.DevPak>

<http://home.wanadoo.nl/m.nealon/wxWidgets-2.6.2contrib.DevPak>

<http://home.wanadoo.nl/m.nealon/wxWidgets-2.6.2contrib.DevPak>

Advanced Users

Advanced users may be interested to know where the devpaks are stored that are installed by PackMan. The answer is in your wxDev-C++ installation directory in the folder called packages. Why is this of interest?

I run three different installations of wxDev-C++, one on my home computer, one on my laptop and one on a virtual vmware install of Windows which I use to test on. Instead of repeatedly downloading and installing the packages, I install a package once on my main computer and then install on the other machines by copying the .devpak files across and then using the [Install] option in Packman.

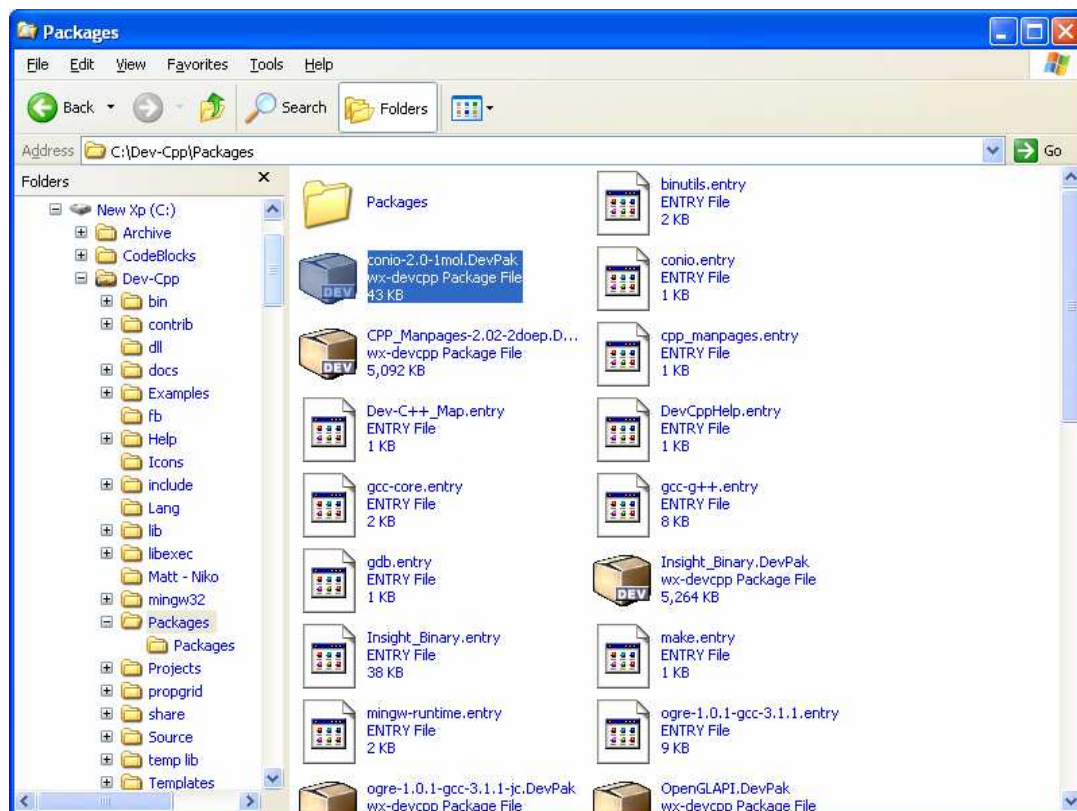


Figure 1.46 – Local versions of the installed DevPaks

Chapter 2 – Compiling your first program

Introduction

So you have your new IDE installed and upgraded as you wish. So what to do with it? This chapter will deal with how to open existing projects and how to create and save your own projects.

Instead of drowning you in screenshots this chapter will start using certain conventions listed below.

Menus

When you see in the text a line like

File|New|Project

it means go to the File menu on the menubar at the top of the IDE. Select 'File' by clicking on it, move down to the option 'New' and select 'Project' from the pop out menu. This is shown below.

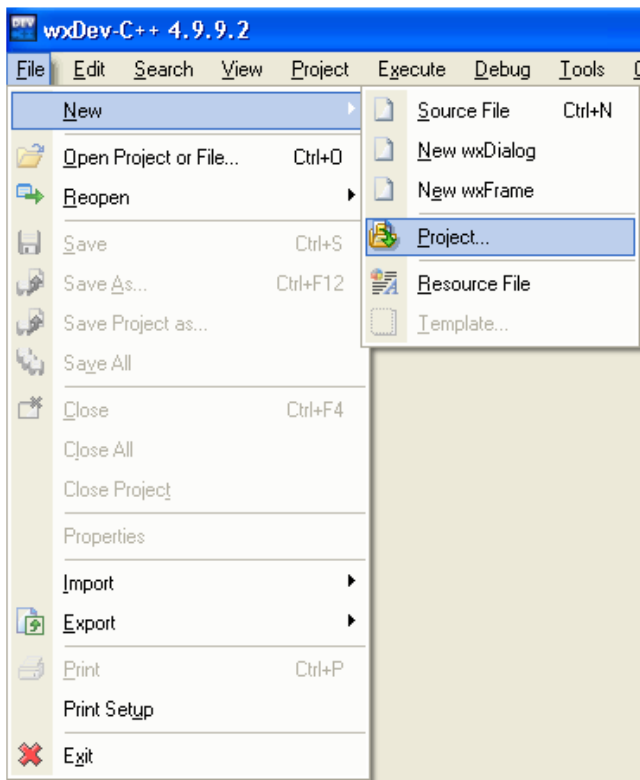


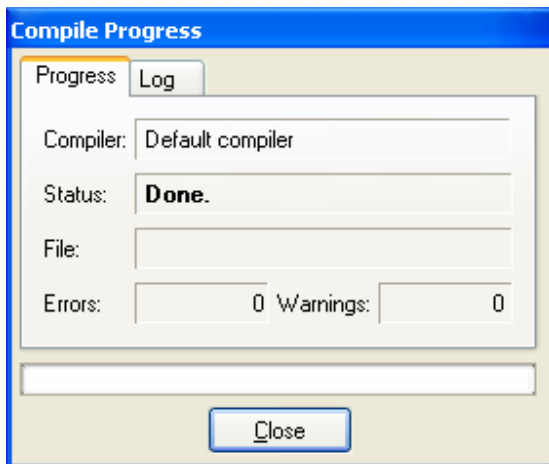
Figure 2.1 – Demonstration of File|New|Project

Keyboard Shortcuts

When you see instructions like press <Ctrl><F9>, this means to hold down the key labelled 'Ctrl' on your keyboard and while holding it press the key labelled <F9>. There are three types of combination keys 'Ctrl', 'Shift' and 'Alt'. Both 'Ctrl' and 'Alt' can be found on the bottom of the keyboard. 'Shift' can be found on the left and right-hand sides of the keyboard. Keys beginning with 'F' can be found on the top row of the keyboard and are known as Function keys. For more information regarding keyboard shortcuts in wxDev-C++ see Appendix 1.

Onscreen Buttons

When you see instructions like press [Close], this means to locate the button onscreen with the text 'Close' on it and click this with your mouse. This is demonstrated in the following screenshot.



Opening an existing project

DevC++, and therefore wxDev-C++, comes with a number of example projects to compile and play with to aid your learning. We will start by opening and compiling one of these projects.

Ensure wxDev-C++ is running. If the tip of the day window is displayed, close it by clicking [Close]. Now go to:

File|Open Project or File

This will open the 'Open File' dialog. Depending on where you last opened a file this dialog will display that directory. This dialog displays differently on other platforms so don't worry if yours looks different to mine.

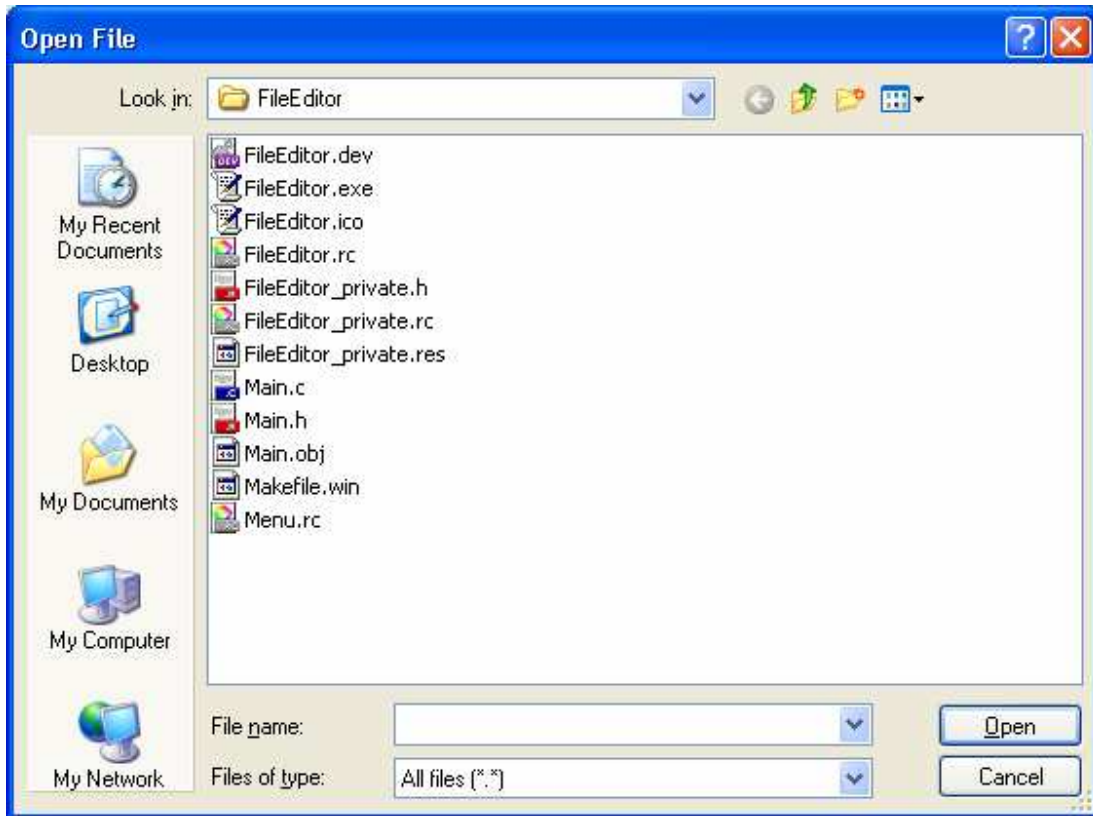


Figure 2.2 – The Open File dialog

The sample we are about to use is stored in the `\Dev-Cpp\Examples` folder so you need to navigate to it either using the folder list box (the one with the 'Look In' prompt) or by using the 'Up One Level' icon. (If you installed wxDev-C++ in the default location then the whole path will be `'C:\Program Files\Dev-Cpp\Examples'`). You should see the following list.

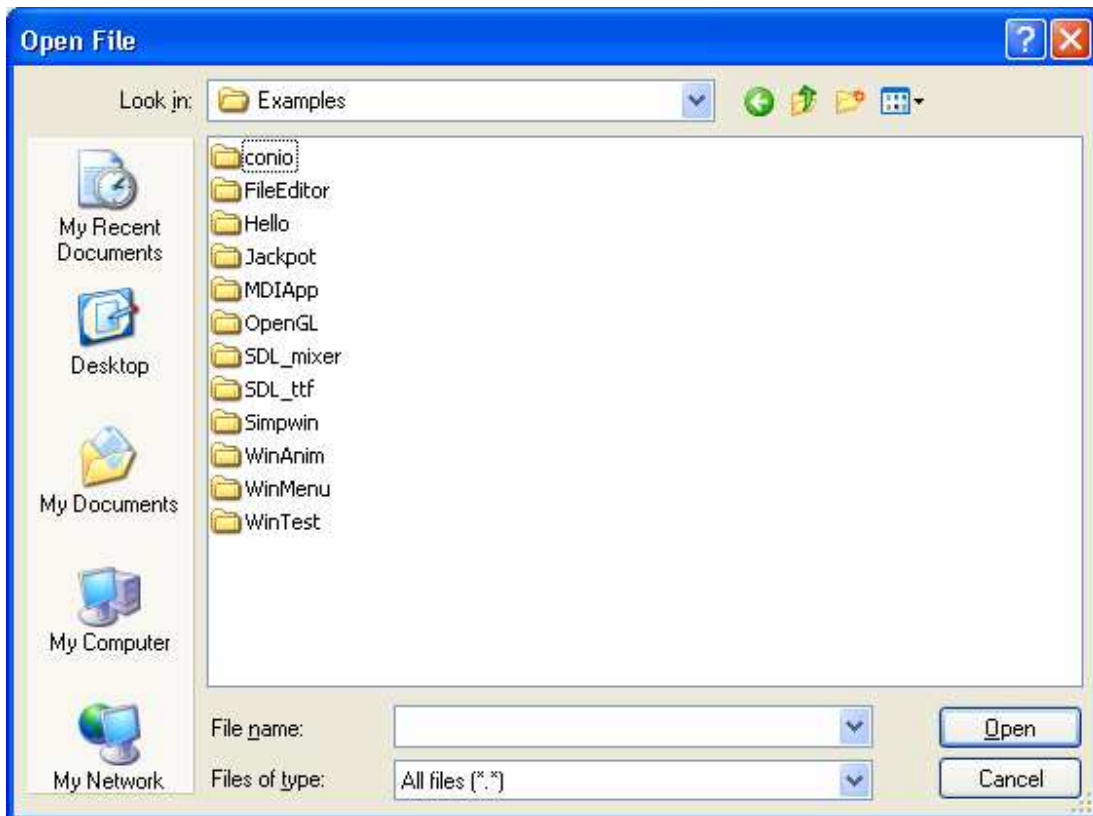


Figure 2.3 – The examples supplied with DevC++/wxDev-C++

Open the folder 'Jackpot' and examine its contents. You should see the following list of files. The one we want to open is called 'Jackpot.dev'. Either:

- double click it to open it
- or select it and press [Open].

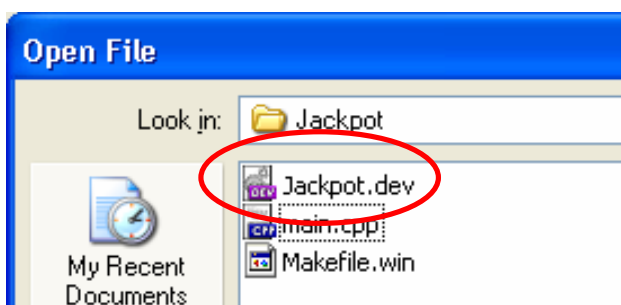


Figure 2.4 – The selecting a .dev project file

The .dev file contains various project settings. This includes things such as the names of the files used in the project, options for the compiler, version numbers etc. Later on you will learn how to alter all the settings which are included in this file.

Now you have opened the .dev file you are returned to the IDE. The tree control on the left displays all the files included in this project, when the 'Project' panel has the focus. For this project there is only one file called 'main.cpp'.

Click on 'main.cpp' to open it in the IDE.

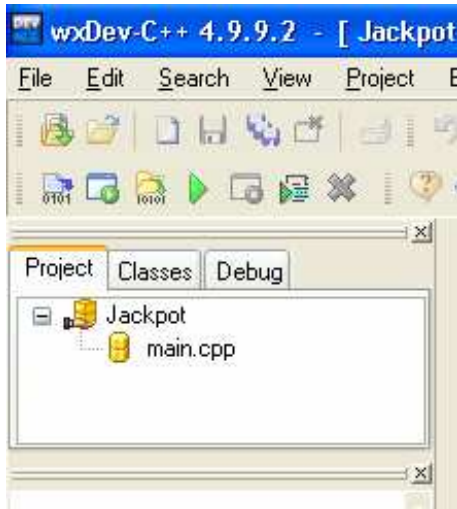


Figure 2.5 – The list of files included in this project

The file will open in the IDE. What you are looking at now is called the 'Source Code'. You will notice that different line and parts of lines are in different colours. This is called 'Syntax Highlighting' and enables you to easily distinguish different parts of the source code at a glance. The colouring used by the syntax highlighter can be configured to suit your preferences so don't worry if your colouring differs from mine.

Looking at the image below you will see the top three lines are coloured **green**. The lines begin with '#' and are known as 'Preprocessor' lines. We will deal with the Preprocessor in more depth later.

Next you will see that certain words are shown in **bold** print. These are 'Reserved Keywords'. Reserved Keywords are words that are part of the programming language and which you cannot use for your own purposes. You will also notice that they are all in lower case. C and C++ are case sensitive languages, so 'Save' and 'save' are different.

Parts of lines beginning and ending with '"' are known as 'String constants' and are coloured **red**. Number constants are shown in **purple**.

Finally lines beginning with '//' or beginning with '/*' and ending with '*/' are coloured **blue**. These are comments. Comments are there for you and other human readers to help understand the meaning of the source code. The compiler takes no notice of comments, so use them more than you think you need to. When you come back to a tricky piece of code in a years time, well placed comments will dictate how long it takes to understand the code.

```
main.cpp
#include <iostream>
#include <stdlib.h>
#include <time.h>

using namespace std;

void Start ();
void GetResults ();

int i, j, life, maxrand;
char c;

void
Start ()
{
    i = 0;
    j = 0;
    life = 0;
    maxrand = 6;

    cout << "Select difficulty mode:\n"; // the user has to select a difficultly level
    cout << "1 : Easy (0-15)\n";
    cout << "2 : Medium (0-30)\n";
    cout << "3 : Difficult (0-50)\n";
    cout << "or type another key to quit\n";
    c = 30;
}
```

Figure 2.6 – Syntax highlighted source code.

We won't spend any time now trying to understand what all this means, because now is the time for you to compile your first program. To compile means to pass the human readable (I promise you will be able to read and understand this later) source code to a program called a compiler. The compiler then translates this into binary code understandable by a computer. Once the programme has finished compiling, providing it finds no errors in it (See Debugging with wxDev-C++), you can run it.

There are a number of ways to compile the program but the quickest is to press <Ctrl><F9> (See the introduction for more details). Alternatively you can use the menu option Execute|Compile or you can press the compile button on the toolbar.



Figure 2.7 – The compile button

The compile dialog will popup next. Depending on the size of your project this next part may take a while, but for this program it will take a second or so. When the compiler has finished the [Cancel] button will change it's caption to [Close].

Click on the [Close] button.

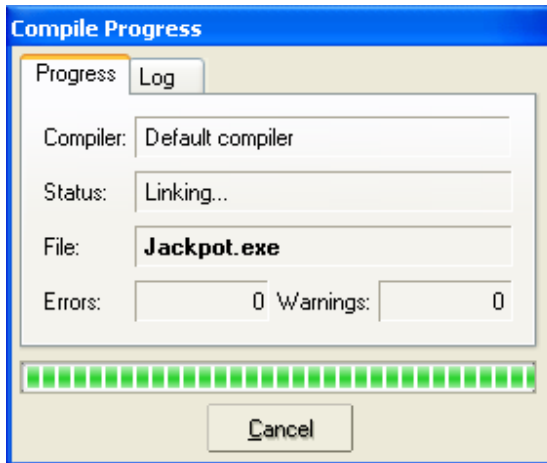


Figure 2.8 – Compiling.

You are now back at the IDE, so what happened? Where is your new program? Don't worry we have only built the programme, so now we need to run it. This can also be done from within the IDE. Once again you have several options:

- Press the keyboard shortcut <Ctrl><F10>.
- Or select 'Run' from the menu Execute|Run.
- Or use the [Run] button on the toolbar.



Figure 2.9 – Running the program

Hey presto, your new program is up and running. Play with it for awhile. The object of the game is to guess the number the computer has chosen between 0 and 30. When you are bored press any key and then <Enter> to exit the programme .

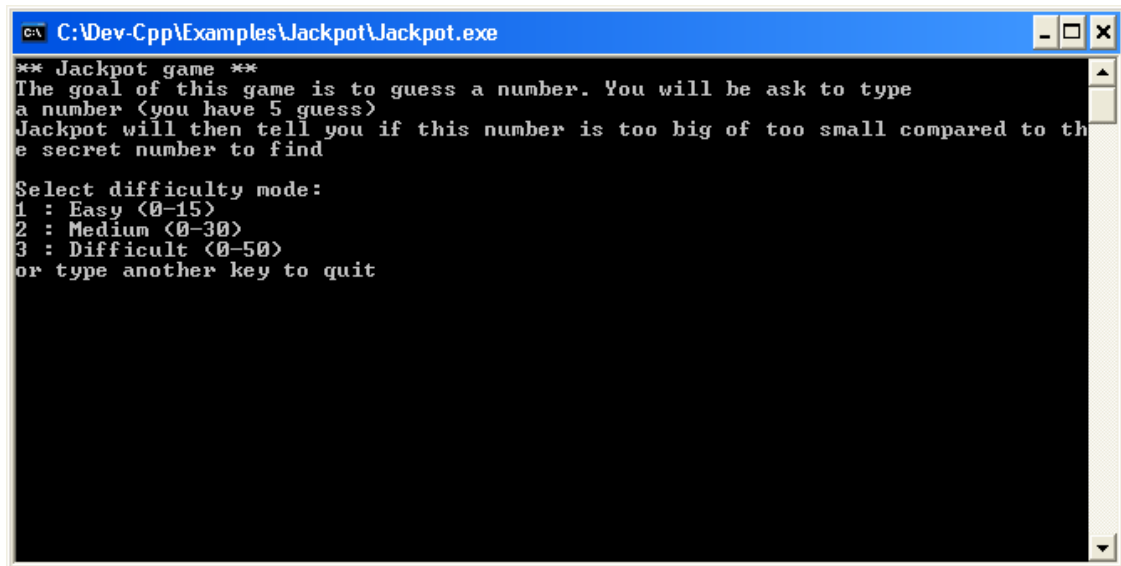


Figure 2.10 – The jackpot game

So far so good, but wouldn't it be nice to compile *and* run the programme all in one go? Well don't be so lazy ☺. But in case you are and all the best programmers are lazy (actually just interested in saving time when doing repetitive tasks). You can compile and run using one of the following methods:

- Press <F9>.
- Or Execute|Compile & Run from the main menu.
- Or use the Compile & Run button on the toolbar.



Figure 2.11 – Compiling and running in one step

Congratulations you have not only learnt to open projects, but also how to compile and run them in one step therefore doubling your productivity. (For the male audience, who said 'Men can't do two things at once?').

Creating your own project

So you have finished playing Jackpot and you are ready to move on. This chapter was called Compiling Your First Program. You have now compiled a programme, but not your own, so let us move on and do just that.

There are two ways of creating a new project:

From the menus select File|New|Project.
Or from the toolbar, select the 'New Project' button.



Figure 2.12 – The new project toolbar button

Either of these methods will provide the New Project dialog. Depending on what packages you have installed on your system this will differ accordingly. You may have more or less tabs and more or less options on each tab.

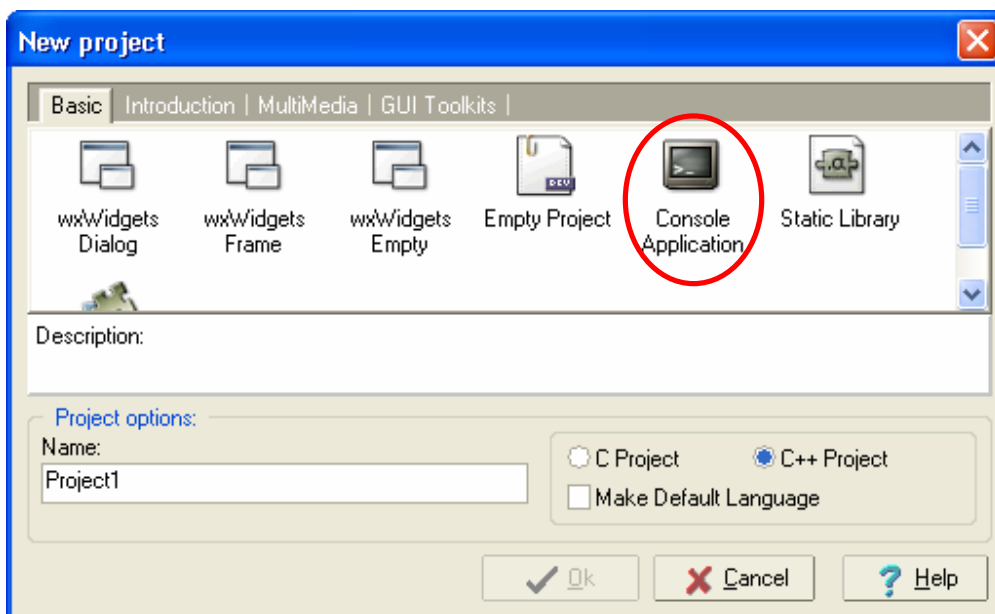


Figure 2.13 – The new project dialog.

Among the options visible on this tab should be 'Console Application'. If you have many options here you may need to scroll down until you find it.

Click on the 'Console Application' icon.

The window labelled 'Description:' should now alter to give you a basic description of this project. In this case it will say 'A console application (MSDOS window)'. The other options shown in this dialog are the project name.

Type 'MyHelloWorld' in the name field
leave the other settings as they are
press the [OK] button.

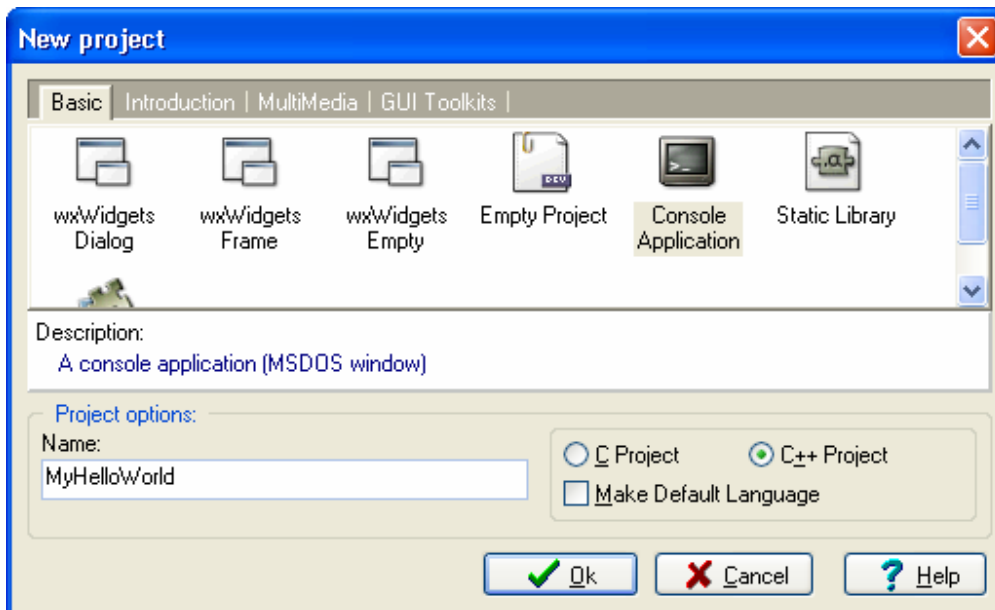


Figure 2.14 – How the New Project dialog should look

If you already have a project open a dialog will popup asking you if you really want to close this project and start a new one. Select 'Yes'. If you have any unsaved files you will be prompted to save them.

Next you will be presented with a dialog asking where to save the project file. Personally I browse to c:\Dev-Cpp and there I create a new folder called Projects (if it doesn't already exist) by clicking on the [Create New Folder] button.

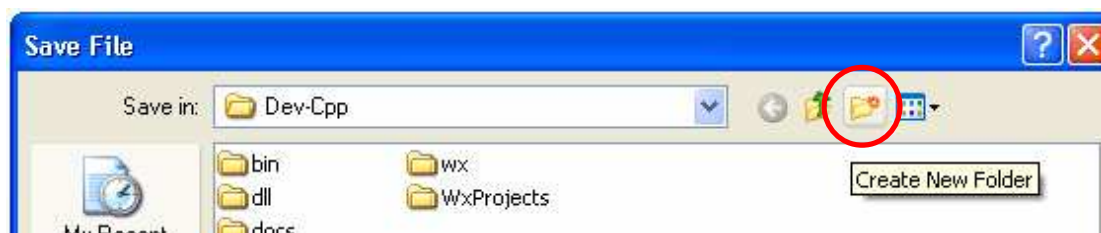


Figure 2.15 – Creating a new folder

The new folder will be created and you will be able to edit the name.

Change the folder name to 'Projects', this folder will become our main store for all future projects.

Access the 'Projects' folder by double clicking the folder name.

Create another new folder, this time call it 'MyHelloWorld'. (For safety's sake don't leave any spaces in the name since Dev-C++ reportedly has problems with spaces in file names.)

Access the "MyHelloWorld" folder by double clicking the folder name.

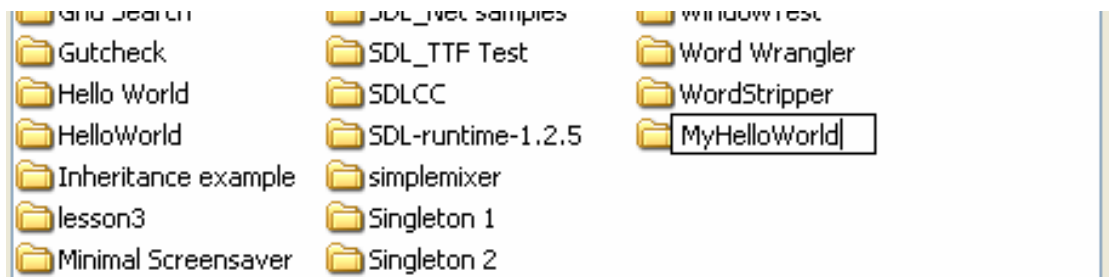


Figure 2.16

The filename is already filled in from the name you chose for the project in this case 'MyHelloWorld.dev' so just press 'Save'.

The project file will be saved and the IDE will display a basic source code file like the following.

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main(int argc, char *argv[])
{
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

Again I will not go into details about what all this means as we cover this in the next two chapters. Instead alter the source code to the following, making sure you change the string constants to your own words.

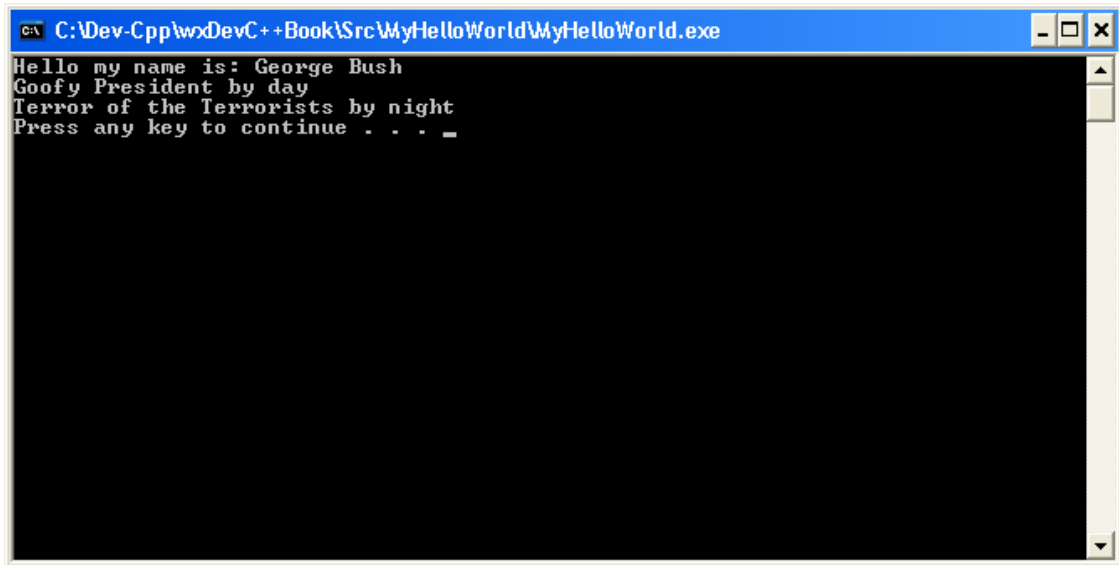
```
#include <cstdlib>
#include <iostream>

using namespace std;

int main(int argc, char *argv[])
{
    //Change the text Your name goes here to your name
    cout << "Hello my name is: " << "Your name goes here" << endl;
    //You can change the text between the quotes to describe yourself by day
    cout << "Mild mannered reporter by day" << endl;
    //You can change the lines between the quotes to describe your super self
    cout << "Caped crusader by night" << endl;
    //This line pauses the program, try putting // in front of it to see what
    //happens
    system("PAUSE");
    //This line says that the program has terminated normally, ie not crashed
    return EXIT_SUCCESS;
}
```

Press <F9> to compile and run your first program.

A pop up dialog will prompt you to save your source code. Check that the directory shown at the top next to the label 'Save in:' is our project directory, in this case, 'MyHelloWorld'. wxDev-C++ will automatically have titled the source code file 'main.cpp'. The '.cpp' extension tells the compiler and us that this is a C++ source code file, not C or any other language. You can change the name (but not the extension) if you wish, but I would leave it as is and press the [Save] button. Immediately the compiler will start and a second or so later the program will run. If you made the changes suggested they will be displayed on the screen.



```
C:\Dev-Cpp\wxDevC++Book\Src\MyHelloWorld\MyHelloWorld.exe
Hello my name is: George Bush
Goofy President by day
Terror of the Terrorists by night
Press any key to continue . . . _
```

Figure 2.17 – Output from MyHelloWorld program.

Congratulate yourself, you have just successfully written and compiled your first program. Welcome to the rank of C++ programmers. But to become more proficient study the next two chapters.

Chapter 3 – Basic C Programming

Introduction

In 1972 Dennis Ritchie wrote the C programming language for use on the unix operating system. In 1978 Kernighan & Richie wrote 'The C Programming Language', this provided a semi standard C known today as K&R C. In 1983 the ANSI (American National Standards Institute) created a standard definition of the C language. This was completed in 1988 and is known as ANSI C. This more or less makes K&R C obsolete although you may come across it from time to time. Since then the ANSI committee have produced a newer standard of C known as C99, the older version being known as C89. C99 adds a few necessary features and borrows a few from C++.

Neither this nor the next chapter are designed to teach you how to design programmes, which is a topic which could fill several more books (and start a few minor wars as well). Neither are they designed to teach you the C and C++ languages in depth. That is the place of a whole shelf full of books.

These two chapters will skim through the basics of these languages and by means of examples give you a slight grounding in these. There are many excellent books on these topics and several excellent websites. (Refer to learning resources in appendix yy)

WARNING: C and C++ are called 'Case Sensitive' languages. This means that 'printf' is not the same as 'PRINTF' or 'Printf'. Other languages are not so strict and this can cause problems for newcomers from such languages.

Break down of a simple example

To start with we shall look at a basic example of a C language source code file. We shall create this in wxDev-C++ and compile it to see what it does. Next we will break it down to see how it does what it does.

To begin the process:

- Ensure wxDev-C++ is open
- Create a new project (as in the previous chapter)
- Select the 'Console Application' option.
- Select the project option 'C Project'
- Name the project 'SimpleC'

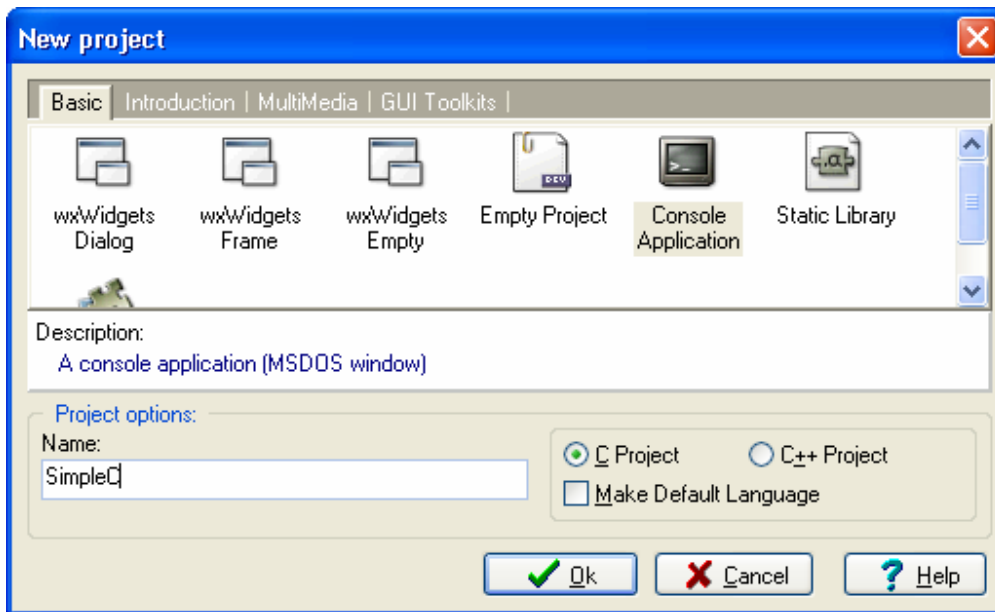


Figure 3.1 – Project settings for the SimpleC program

Click the [OK] button and you will be prompted to close any open projects, then to save this project:

Browse to the Project folder we created earlier and create a new folder inside it called “SimpleC”.

Open the 'SimpleC' folder and save the project file.

You will be returned to the IDE with the following freshly generated code

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
    system("PAUSE");
    return 0;
}
```

Firstly to make it a little easier to know which lines I am referring to, let us turn on the line numbering feature in wxDev-C++.

Select the main menu option: Tools|Editor Options.

On the Editor Options dialog:

Click on the second tab labelled 'Display'
Click on the check box next to 'Line Numbers'

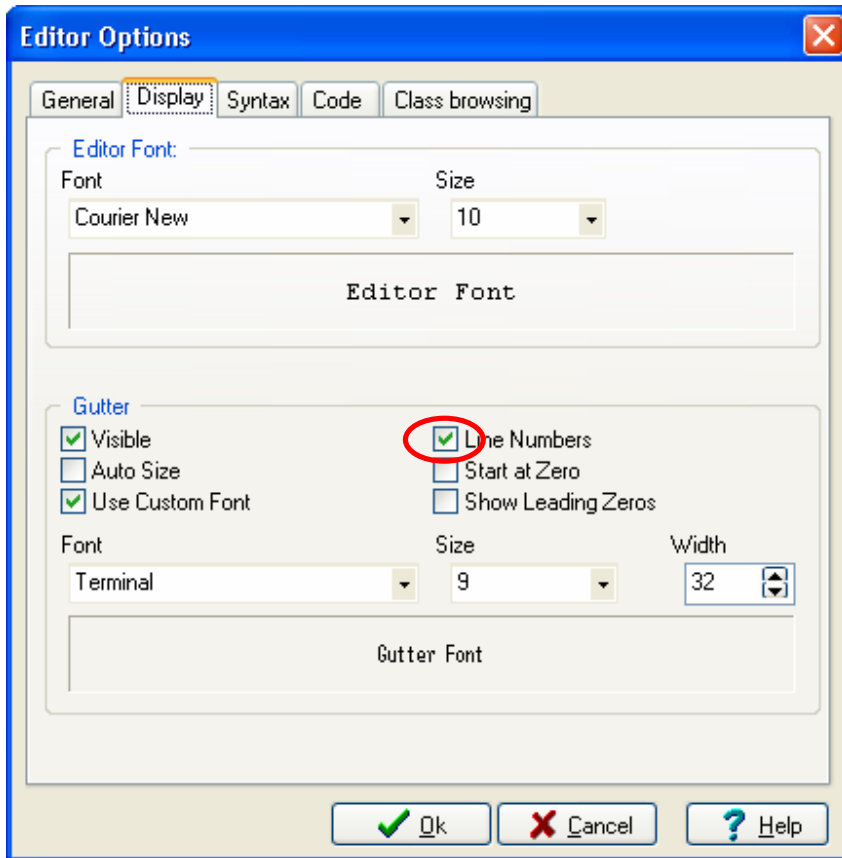


Figure 3.2 – Displaying line numbers

Alter the code to match the following

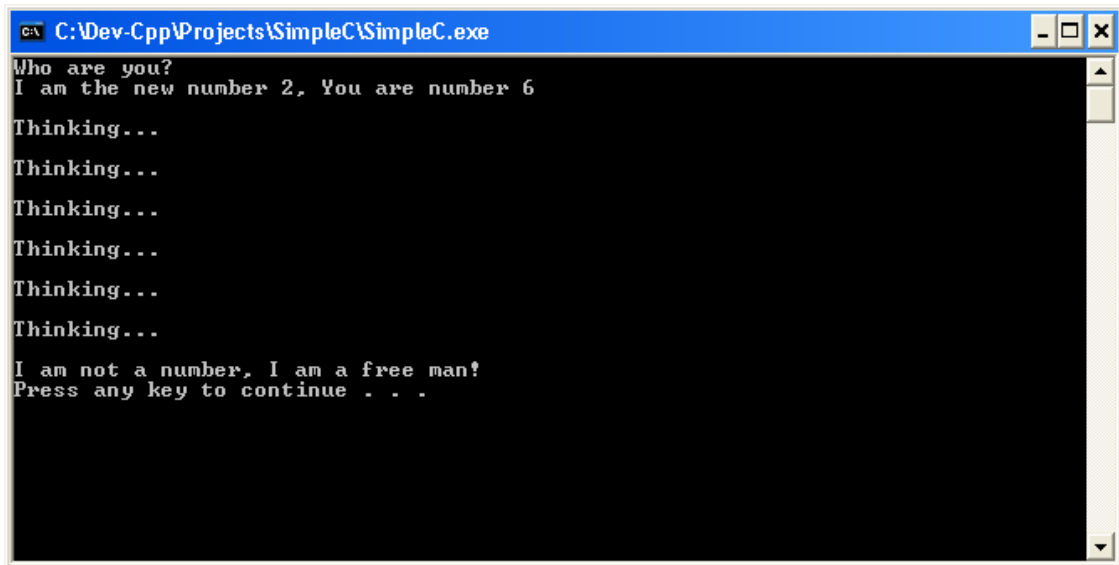
```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  #define TOP_MAN 2
5  #define THE_PRISONER 6
6
7  int main(int argc, char *argv[])
8  {
9      int i = 0;
10
11     printf("Who are you?\n");
12     printf("I am the new number %d, ", TOP_MAN);
13     printf("You are number %d\n\n", THE_PRISONER);
14
15     for(i = 0; i < THE_PRISONER; i++)
16     {
17         printf("Thinking...\n\n");
18     }
19
20     if(i == THE_PRISONER)
21         printf("I am not a number, I am a free man!\n");
22
23     system("PAUSE");
24     return 0;
25 }

```

Press <F9> to compile and run the program.

You will be prompted to save the source code file, save it in the same directory as the project file. You should be greeted with the following output.



```
C:\Dev-Cpp\Projects\SimpleC\SimpleC.exe
Who are you?
I am the new number 2, You are number 6

Thinking...
Thinking...
Thinking...
Thinking...
Thinking...
Thinking...
Thinking...

I am not a number, I am a free man!
Press any key to continue . . .
```

Figure 3.3 – Output from the SimpleC program

I am now going to break this example down into various parts which I will discuss briefly here and then in greater depth in the following relevant sections.

The C programming language has only 32 keywords (listed in Appendix B). None of which include the provision to output to the screen. Before you say ‘But wait a minute, I just wrote a program which output a lot of nonsense to the screen’, let me explain. What C lacks in keywords it makes up for in libraries. ANSI C has a large number of libraries containing functions of various purposes. The function `printf` is one of these.

Before we can use these functions we need to tell C that we wish to make use of the library containing that function. We do this using `#include` statements. We can see examples of this in lines 1 & 2. All lines beginning with `#` are called ‘Preprocessor Lines’ and we will deal with these in the section ‘Preprocessor’. There are also two more examples in lines 4 & 5, these lines begin with `#define` and create constant values.

As we have mentioned, C is made up of functions. Later on, in the section ‘Functions’, you will create your own functions to get a better feel for them. C demands that there is at least one function present in every programme and that is the `main` function. A function consists of a function header and a body section enclosed by curly braces `{}`. The code enclosed within the braces on lines 8 & 25 are in what is known as the ‘body’ of the function.

Values can be stored in variables, there is one declared on line 9. The first part **int** is a keyword and tells us that the variable will store an integer. An integer can only contain whole numbers. We give the variable a name 'i' so we can refer to it later, as in lines 15 & 20.

WARNING: In C when a variable is created it can contain any value, so it is best to assign it a known default such as line 9 where 'i' is assigned the value '0'. Again this is a pitfall for new programmers or those from other languages.

Lines 15 to 18 are an example of a 'Control Loop' and will be covered in the next section 'Basic C' as will lines 20 & 21 which contain an example of a 'Conditional Execution'.

The other lines will be covered in 'Input/Output and other useful functions'.

Basic C

To store data within a C program we use variables. These are defined as in line 8 by first stating the type of data the variable is to hold, then assigning the variable a unique name. It is good form to initialise the variable by assigning it a value at this point.

Data types

There are 5 basic data types in C89. These are `char`, `int`, `float`, `double` and `void`. `char` is designed to hold data values corresponding to a character. `int` is designed to hold whole numbers. `float` and `double` hold floating point numbers and `void` is a null value. Some of these data types can be extended with the keywords `signed`, `unsigned`, `long`, and `short`. `int` can take all of these values, `char` can be `signed` or `unsigned` and `double` can have `long` applied. Unsigned data types can only hold numbers from 0 to their maximum range. Signed data halve this and give half the range to negative figures and half to positive figures. The C standard specifies a minimum range for data types, but not a maximum which can vary between compilers and platforms. The following program called `DataSize` demonstrates these sizes.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main(int argc, char *argv[])
5  {
6      printf("The size of a char is          %d bits\n", 8 * sizeof(char));
7      printf("The size of a unsigned char is %d bits\n", 8 * sizeof(unsigned
char));
8      printf("The size of an int is          %d bits\n", 8 * sizeof(int));
9      printf("The size of a unsigned int is  %d bits\n", 8 * sizeof(unsigned
int));
10     printf("The size of a long int is      %d bits\n", 8 * sizeof(long int));
11     printf("The size of a short int is     %d bits\n", 8 * sizeof(short int));
12     printf("The size of a double is        %d bits\n", 8 * sizeof(double));
```



```
13 printf("The size of a float is      %d bits\n",8 * sizeof(float));
14 printf("The size of a void is      %d bits\n",8 * sizeof(void));
15 printf("The size of a long double is %d bits\n",8 * sizeof(long double));
16 system("PAUSE");
17 return 0;
18 }
```

Variable names

Variable names have certain limitations. The name cannot be a keyword, it must be unique and can contain any alpha numeric characters and underscores, as long as the name does not begin with a number.

Statement blocks

You will have noticed in the code examples so far, the use of ‘{ }’ braces. These braces enclose code within what is called ‘Statement Blocks’. The significance of this can be seen in the next section ‘Scope’. Braces must always be used in pairs, i.e. the opening brace { must have a corresponding closing brace }. If you have problems with mismatched braces see the warning on under *parenthesis operators*.

Scope

Variables are said to have ‘Scope’ in C and you can only refer to or use variables that are in scope. There are several types of scope. One is global scope, variables of this type can be accessed by all parts of your programme (which may involve several source files). A second type is file scope, variables of this kind can only be accessed by code within a single source code file. The significance of the difference between global and file scope will be apparent later. The last type that we will discuss is local scope. Variables declared within statement blocks are only visible to code that is also contained within the same block.

Operators

Operators (=,&,%,! ,| etc) break down into several groups: assignment, arithmetic, relational, logical, bitwise, pointer, reference, indexing and parenthesis are all operator types. There are a few others which we won't discuss at all. Pointer operators will be discussed under ‘Pointers’, reference operators will be discussed under ‘Structures’ and indexing operators under ‘Arrays’. Bitwise operators will not be discussed at all.

Assignment Operators

The main assignment operator is ‘=’ for example `int x = 0;` This expression assigns the value zero to the variable x. The value being assigned must always be on the left of the expression, eg `0 = int x` is not a valid statement. It is also possible to do multiple assignments for example `x = q = r = t = 0;` In this example the variable t is assigned the value zero, then the variable r is assigned the value of t and so on down to

x. It is also possible to declare and assign multiple variables of the same type at the same time eg `int i=0, a=5,b=10,c=15;`

If you assign variables of one data type to variables of another data type, then what is known as an ‘automatic type conversion’ takes place. When the value on the right side of the expression is assigned to the variable on the left, an attempt is made to automatically convert it to the data type of the left side variable. If the type on the left has a smaller range than the type on the right then data will be lost. The following example program demonstrates this.

Create a new project as normal.
Call it ‘TruncationAssignment’.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main(int argc, char *argv[])
5  {
6      short int aShortInt= 0;
7      float aFloat = 536546.64645;
8      long int aLongInt= 2045654297;
9      /*Print the value of the long int*/
10     printf("The long int value is: %d\n",aLongInt);
11     /*Convert the long int to a short int*/
12     aShortInt = aLongInt;
13     printf("The long int has been assigned to a short int\n");
14     /*Print the value of the short int*/
15     printf("The value now = %d\n\n",aShortInt);
16     /*Repeat for a float to int*/
17     printf("The float value is: %f\n",aFloat);
18     printf("The float has now been assigned to a short int\n");
19     printf("The value is now: %d\n",aShortInt);
20     system("PAUSE");
21     return 0;
22 }
```

The compiler may warn you about the possibility of data loss when you do this. If you intend to convert from one data type to another you can tell the compiler that you intend this by using `cast`. For example if `x` is an integer then you can use this expression `float y = (float) x/3;`

There are also shorthand assignment operators. Sometimes you will want to assign a variable to itself plus or minus another value. For example `x = x - 20;` you can do this in shorthand using `x -= 20;` ‘+=’ is also equally valid. You will see this use of shorthand operators a lot in professional programs.

Arithmetic Operators

There are seven different arithmetic operators, the basic four:

+ for addition
/ for division

- * for multiplication
- for subtraction

There are two shorthand arithmetic operators. These are -- and ++ which are used to increment or decrement a variable by one. For example you can write `x = x + 1;` or `x++;`. The shorthand operators can be used before or after a variable, for example `y = x++;` or `y = ++x;`. You may ask what is the difference? The difference is that the first one sets y to equal x then sets x to equal x + 1. The second sets x to be x + 1 then assigns the result to y. A subtle but important difference.

The final operator is the modulus operator %. This can only be used with integers and it returns the remainder of an integer division.

Relational Operators

Relational operators compare two different values and give a result of true or false (1 or 0). The six relational operators are:

- > greater than
- >= greater than or equal to
- < less than
- <= less than or equal to
- == equal to
- != not equal to

Logical Operators

Logical operators can be used with true or false (that is 1 or 0) values. There are three such operators:

- && AND
- || OR
- ! NOT

They give the following values according to this table.

X	Y	X && Y	X Y	!X
0 (false)	0 (false)	0 (false)	0 (false)	1 (true)
0 (false)	1 (true)	0 (false)	1 (true)	1 (true)
1 (true)	1 (true)	1 (true)	1 (true)	0 (false)
1 (true)	0 (false)	0 (false)	1 (true)	0 (false)

Because logical operators work with true/false values they can be combined with relational operators. For example `4 > 6 && 3 < 7 || 7 != 5`.

NOTE: There is no XOR operator in C, but this can easily be created using a function.

Parentheses Operators

C operators work on the order of precedence. This means that in an expression like the following $x = 40 + 3 / 5 + 7 * 352$; does not mean that you can evaluate it from right to left. Depending on the precedence of operators the order of calculation may be different. To force this to be evaluated in the way you want you need to use parentheses operators '()'. For example $x = (40 + (3 / (5 + (7 * 352))))$. The innermost pair of brackets is calculated first and then the order works outwards from that.

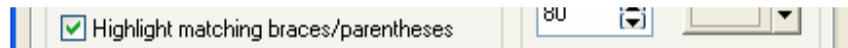
WARNING: Make sure that you have the same number of closing brackets as opening brackets otherwise the compiler will issue a complaint like this:

```
main.c: In function `main':  
main.c:6: error: syntax error before ';' token
```

equally if you have more closing brackets than opening brackets you will get a complaint like the following:

```
main.c: In function `main':  
main.c:6: error: syntax error before ')' token
```

If you are having trouble with mismatched braces you can turn on brace highlighting. Go to Tools|Editor Options. On the dialog that appears check the box next to 'Highlight matching braces/parentheses' as shown below.



Now if you select a bracket the editor will attempt to show you the matching bracket.

Conditional Loops

Left to its own devices, a program starts at the top and works through to the end where it quits. Sometimes between starting and finishing you want the program to carry out the same sequence of commands again and again. For instance you might want to print out "Go away" 20 times until someone gets the message. You could just type in the commands one after the other like this.

```
printf("Go away!\n");  
printf("Go away!\n");  
  
...  
  
printf("Go away!\n");  
printf("Go away!\n");
```

This might not seem too much of a hardship, but what if you wanted to do it 2000 times, you would soon get bored of writing the same thing. It would also be hard to maintain the code. What if you later had to reduce it to 863 times? Fortunately there is a much simpler method called 'Conditional looping'.

C has 3 types of loops the `for` loop, the `while` loop and the `do...while` loop. The `do...while` loop guarantees that the contents will be executed at least once since the loop condition is not checked until the end. The code contained in `while` and `for` loops may never be executed since the condition is checked at the beginning of the loop, and if it is not true the loop statement is skipped.

The `for` statement has this general form:

```
for( <loop initialization>,<condition check>,<loop increment>)
```

It is possible to make use of the 'for' statement in many unusual ways, but we will only consider the basic usage here. The 'while' and 'do...while' loops are much simpler since they contain only the condition check.

Create a new project called 'Loopy' and amend the code to the following.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main(int argc, char *argv[])
5  {
6      /* Variable to check the loop condition */
7      int i = 0;
8      printf("For loop countdown to liftoff\n");
9      for(i=10;i>0;i--)
10     {
11         printf("%d\n",i);
12     }
13     printf("We have lift off.\n\n");
14     system("PAUSE");
15
16     printf("\nWhile loop count to five\n");
17     /*Reset i to 0, then try a while loop*/
18     i = 0;
19     while(i <= 5)
20     {
21         printf("Just checked condition i <= 5 and ");
22         printf("i actually = %d\n",i);
23         /*Need to increment i*/
24         i++;
25     }
26     system("PAUSE");
27
28     printf("\nDo...while loop count to five\n");
29     /*Reset i to 0, then try a do...while loop*/
30     i = 0;
31     do
32     {
```

```

33     printf("i = %d and ",i);
34     /*Need to increment i*/
35     i++;
36     printf("just about to checked condition i <= 5\n");
37     }while(i <= 5);
38     system("PAUSE");
39
40     return 0;
41     }

```

Code analysis: On line 7 we create the counter variable `i` and initialize it to the value 0. In lines 9 to 12 we create a `for` loop. Above we have shown that the `for` statement can take three arguments, the initialization, the condition check and the increment.

The initialization `i=10` sets our variable `i` to 10 and is only done once.

The condition check `i>0` checks that `i` is greater than zero and is done on each loop. When the condition check produces a false answer, the loop ceases.

The increment `i--` decreases the value of `i` by 1 on each iteration through the loop

In lines 19 to 25 we create a `while` loop. The condition is checked before the loop runs, and must remain true for the loop to continue running. In this case that `i` is greater than or equal to 5. Line 24 increases `i` by one during each loop.

In lines 31 to 37 we create a `do...while` loop. This is very similar to the `while` loop, except that the condition is checked at the end.

If the `for` loop contains just a single line of code then you can use the shorthand form which leaves out the brackets. For example lines 9 to 12 could have been written as

```

9         for(i=10;i>0;i--)
10            printf("%d\n",i);

```

In this case the code executed by the `for` loop ends at the semicolon at the end of line 10. It is possible to escape from a loop at anytime by using the keyword `break`, this will leave the loop and begin executing the code directly after the loop.

NOTE: The `while` loop ends on line 25 with a `}` bracket. However the `do...while` loop ends on line 37 with `while() ;`. The semicolon at the end of the `while` statement is vital, if you miss it out the program will not compile.

WARNING: All three loops altered the check condition by altering the value of `i`. If the value of `i` had not changed the check condition would always be true and the loop would continue for ever. To check this, delete line 35 and compile and run the program again. To stop the program press `<Alt><F2>` from within `wxDev-C++` or click on the button shown.



Conditional Execution

In programming, as in life, there will be times when you want to do something and times when you don't. For example if it is raining you will probably want to stay in bed and sleep. If it is sunny you probably won't. This is called 'Conditional execution' and can be achieved using the 'if' statement.

The if statement takes the form:

```
if(<Check Condition>)
```

Sometimes you want to do several different things depending on the value of the check statement, in this case you can follow the if check with else if checks or an else to do something in the event of an if check failing. Example code follows, so create a new project and call it 'IfBob', then modify the code to look like the following.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main(int argc, char *argv[])
5  {
6      int BobsAge = 0;
7      for(BobsAge = 0;BobsAge <= 100;BobsAge += 10)
8      {
9          if(BobsAge <= 0)
10         {
11             printf("Bob has not been born yet\n");
12         }
13         else if (BobsAge <= 10)
14         {
15             printf("Bob is a young boy\n");
16         }
17         else if (BobsAge > 20 && BobsAge <= 30)
18         {
19             printf("Bob is a young man\n");
20         }
21         else if (BobsAge <= 80)
22         {}
23         else if (BobsAge > 80 && BobsAge <= 90)
24         {
25             printf("Bob is an old man now\n");
26         }
27         else
28         {
29             printf("Bob has just got a telegram from the Queen\n");
```

```

30     }
31     }
32     system( "PAUSE" );
33     return 0;
34 }

```

In a group of `if else` statements, each condition is checked in turn from top to bottom until a true one is found. The first true condition is executed. If none of them are true, the code following the `else` statement is executed.

In the example the `'if..else if..else'` statements are inside a `for` loop to make sure that they are all executed. As we noted earlier with `for` statements if a single line of code follows the check then the brackets can be omitted.

The `if` statement is not the only conditional execution statement. There is a shorthand form, (sometimes called a ternary operator since it takes three arguments) `'?:'` which takes the form:

`<Conditional Check>?<Code If True>:<Code If False>`

The following short program illustrates this. Create a new project called `BobsLife`. Change the code to the following.

```

1     #include <stdio.h>
2     #include <stdlib.h>
3
4     #define ALIVE 1
5     #define DEAD 0
6     int main(int argc, char *argv[])
7     {
8         int BobStatus = ALIVE;
9         printf( "Bob is " );
10        BobStatus == ALIVE?printf( "alive\n" ):printf( "dead\n" );
11        system( "PAUSE" );
12        return 0;
13    }

```

Line 10 checks `BobStatus` against the value `ALIVE`. If this is true the first `printf` statement is executed, if not the second is executed.

The final form of conditional execution statement is the `switch...case` statement. This works like a series of `'if..else if..else'` statements. The `switch` statement can only check integer values. The `switch` statement contains an expression, the value of which is checked against a list of case statements. If any of the case statements match the code contained in the `switch` statement, that case statement is executed. Each case statement ends with a `break` statement; if this is omitted then the code in the following case statement is executed. The final statement is called `default`. This contains code to be executed if none of the case statements return true.

The following program will demonstrate this. Create a project called DayCalculator. Modify the code to match the following.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  #define MONDAY 1
5  #define TUESDAY 2
6  #define WEDNESDAY 3
7  #define THURSDAY 4
8  #define FRIDAY 5
9  #define SATURDAY 6
10 #define SUNDAY 7
11 #define NO_DAY 8
12
13 int main(int argc, char *argv[])
14 {
15     int day = MONDAY;
16     for(day = MONDAY; day <= NO_DAY; day++)
17     {
18         switch(day)
19         {
20             case MONDAY:
21                 printf("It's Monday, off to work\n");
22                 break;
23             case TUESDAY:
24                 printf("It's Tuesday, long week ahead\n");
25                 break;
26             case WEDNESDAY:
27                 printf("It's Wednesday, halfway there\n");
28                 break;
29             case THURSDAY:
30                 printf("It's Thursday, one more day to go\n");
31                 break;
32             case FRIDAY:
33                 printf("Thank Crunchie, it's Friday\n");
34                 break;
35             case SATURDAY: /*This is an example of a drop through*/
36             case SUNDAY:
37                 printf("It's the weekend, let's live it up\n");
38                 break;
39             default:
40                 printf("Hey! This day cannot exist\n");
41         }
42     }
43     system("PAUSE");
44     return 0;
45 }
```

To me at least, the switch looks a little strange and 'un' C like. Watch those 'break' statements, they can be the cause of major problems. Other languages like Visual Basic have similar statements such as 'Select'. But in those languages the 'break' statement is not required and the following statement is not executed by dropping through as in line 35 in the above example. If in your program several statements seem to be executed, make sure you have added the 'break' statements. Remember that the default statement will only be executed if none of the others are, or if the preceding 'break' statement is forgotten.

You may wonder why 'C' is designed this way after all would it not just be simpler for execution to stop at the start of the next 'case' statement. Doing this would remove the advantage of being able to execute the same code for more than one case such as in lines 35 to 37 where both Saturday and Sunday require the same output.

NOTE: You may come across statements like `if(x)`, what does this mean. Since an `if` statement checks for true/false values if `x` equals 0 the `if` check will fail, anything higher than 0 is considered to be true. This could be written `if(x != 0)`, but this is considered bad form.

WARNING: C and C++ are unusual in that to assign a value to a variable you use '=' as in `MyAge = 243;`. To test for equality you need to use '==' as in `if(MyAge == 243)`. This is a feature that can cause many errors, especially since this code `if(MyAge = 243)` will compile without the compiler telling you there is an error. This `if` statement will always be true and the code after it will always be executed.

Preprocessor

The preprocessor is an unusual feature of C. It runs before the compiler and alters your source code by inserting function declarations from header files, expanding macro definitions, and determining which code should or should not be compiled. Preprocessor statements always begin with a '#' and unlike C the lines don't end with a semi-colon.

The most common preprocessor is the `#include` statement. This takes the form of either `#include <filename.h>` for library header files or `#include "filename.h"` for your own header files. As you will learn in the next section 'Functions', C does not allow a programmer to use a function until it knows what the function name is, what parameters it should expect to receive, and what value the function will return. `#include` statements add all that information to the start of your source code file. In the SimpleC example we include the files `stdio.h` and `stdlib.h`. We need `stdio.h` for the function `printf`, and `stdlib.h` for the function `system`.

NOTE: On platforms other than Windows include file names are case sensitive and a common error is to use the incorrect case, which results in the compiler not finding the correct file.

The second most common is the `#define` statement. This is used to define a constant name which has a value. Wherever that constant name appears in your code the preprocessor will swap it for it's value. This is often used for constant variables (variables which don't change their value). A useful feature of this is it helps to make your code self documenting. For example you can create a define like this:

```
#define DAYS_IN_A_WEEK 7
```

Then use it in your code:

```
for(i = 0;i <DAYS_IN_A_WEEK;i++)
```

which makes it much easier for others to work out what you intend the code to do, than the line:

```
for(i=0;i<7;i++)
```

It is convention to write constant variables in uppercase.

Along with `#define` are `#ifdef`, `#ifndef`, `#else`, `#endif` and `#undef`. These can be used to include or exclude blocks of code based on the value of a `#define` statement. Code following `#ifdef FRED` will only be compiled if `FRED` has been defined. Code following `#ifndef FRED` will only be compiled if `FRED` has not been defined. `#endif` ends the code block and `#undef` undefines a previous `#define` constant.

Finally in a similar vein are the statements `#if`, and `#elif`, they test for the truth of a statement and work in the same manner as `#ifdef` including code if the statement is true.

The following code example uses all these preprocessor statements. Create a new 'Console Application' called 'Prepro'. Save it in a folder called 'Prepro' and then edit it to resemble the following code example. See if you can work out what the output will be before you compile and run it.

```
1  #include <stdio.h> /*include this library to allow us access to printf*/
2  #include <stdlib.h> /*include this library to allow us to use system*/
3
4  #define RED_BALLOONS 99 /*Define RED_BALLOONS to be equal to 99*/
5  int main(int argc, char *argv[])
6  {
7  #ifdef RED_BALLOONS /*Check if RED_BALLOONS has been defined*/
8      printf("There were %d red balloons\n",RED_BALLOONS);
9  #else /*Do this if the previous #ifdef was false*/
10     printf("There were no red balloons\n");
11 #endif /*End this conditional block*/
12
13 #if RED_BALLOONS < 98 /*Check if RED_BALLOONS is lower than 98*/
14     printf("There were less than 98 red balloons\n");
15 #elif RED_BALLOONS > 98 /*If previous #if was false check this one*/
16     printf("There were more than 98 red balloons\n");
17 #endif /*End this conditional block*/
18
19 #undef RED_BALLOONS /*Remove define RED_BALLOONS*/
20     printf("\nBIG BANG!\n\n");
21
22 #ifndef RED_BALLOONS /*Check if RED_BALLOONS has not been defined*/
23     printf("There were no red balloons\n");
24 #else /*Do this if the previous #ifndef was false*/
25     printf("There were %d red balloons\n",RED_BALLOONS);
26 #endif /*End this conditional block*/
```

```
27
28     system( "PAUSE" );
29     return 0;
30 }
```

Functions

Functions are self contained blocks of code, they can be thought of as mini programs that take in various parameters, do something with these and return a result. Functions are defined like this:

```
Return_Type Function_Name (Parameter_Type Parameter_Name, ...)
{
    ...
    Block of code;
    ...
}
```

The return type is either a data type such as `int`, `float`, etc, a pointer to a memory location, a user defined data type such as a `struct`, or `void`. In the case of `void` you are specifying that the function will return nothing. The function ends when it reaches a `return` statement, (or the end of the code block if the return type is `void`). Any code after a `return` statement is called 'Unreachable code' since it will never be executed.

The function name can be anything you wish with a few caveats. The name must be unique, therefore you cannot create a function called `printf` if you `#include "stdio.h"`, since this would confuse the compiler. The name must also conform to the same naming conventions as variables. It is good form to give the function a name that indicates its purpose, e.g. `DisplayMenu` instead of `MyThingyFunction`.

A function can take any number of parameters or none. Each parameter must be given a name and associated data type. These data types are the same as those previously discussed. Within the function, the parameters act as variables with local scope.

In your C programs you will always use at least one function, this is the `main` function. This is a special function since program execution will start at the beginning of this function and finish at the end of it. `main` follows the same rules as other functions, it has a return type `int`, which is used to indicate whether the program executed without error or not. It also takes parameters, these are passed to the program when it starts.

Functions can serve two different purposes. Firstly they can break your program into smaller subunits to make them easier to understand. Secondly they allow you to reuse blocks of code again and again rather than repeatedly writing the same code block.

We will create a program which uses a couple of basic functions to get a feel for them.

Ensure `wxDev-C++` is running.
create a new 'C' project called 'Functions'

and save it in a folder called 'Functions' in your 'Project' folder.

The code follows below.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  /*This function has no parameters and returns nothing*/
5  void DisplayMenu()
6  {
7      printf("\n1. Calculate 5 + 4\n");
8      printf("2. Print 3 Names\n");
9      printf("3. Quit this lunacy\n");
10 }
11
12 /*This function has two parameters both are integers it adds
13 these together and returns a integer which is the result*/
14 int Calculate(int FirstNumber, int SecondNumber)
15 {
16     return FirstNumber + SecondNumber;
17 }
18
19 /*This function has no parameters and returns nothing*/
20 void DisplayNames()
21 {
22     printf("\nThree Names\nHumphrey\nIngrid\nPeter\n");
23 }
24
25 int main(int argc, char *argv[])
26 {
27     int UsersChoice = 0;
28
29     while (UsersChoice != 3)
30     {
31         /*Call function to show menu*/
32         DisplayMenu();
33         /*Function from stdio.h which reads the users input*/
34         scanf("%d",&UsersChoice);
35
36         if(UsersChoice == 1)
37             /*Call the Calculate function from within the printf function*/
38             printf("\n5 + 4 = %d\n",Calculate(5,4));
39         else if (UsersChoice == 2)
40             /*Call the DisplayNames function*/
41             DisplayNames();
42     }
43
44     return 0;
45 }
```

The programme starts at the beginning of the main function. It declares a variable `UsersChoice` and sets it's value to 0. It then enters a conditional loop, and as long as the variable `UsersChoice` doesn't equal 3, it will continually execute the statements inside. Within the loop, the first call is to the function `DisplayMenu`. This prints the menu on the screen. Then it calls a function that is new to us, `scanf`. (We will learn more about `scanf` later.) The programme now enters a conditional execution area, where, depending on the value of `UsersChoice`, it either calls the function

Calculate or DisplayNames. Finally, once UsersChoice equals 3, the return statement is executed and, as we have learnt, that is the end of the main function so the programme ends.

Does it matter where we put a function in our source code? The answer is both yes and no. To demonstrate this we will create another project.

Carry out the usual project creation process calling the new project PostFunction. Create a new folder called PostFunction and save the project there.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main(int argc, char *argv[])
5  {
6      DisplayBoringMessage();
7      system("PAUSE");
8      return 0;
9  }
10 /*This is a boring function*/
11 void DisplayBoringMessage()
12 {
13     printf( "This is a boring message\n");
14 }
```

Before trying to compile this programme, try to work out what will happen when it runs. Then press <F9>. You should find the programme doesn't run and inside the message window on the bottom of the IDE, displays the following errors.

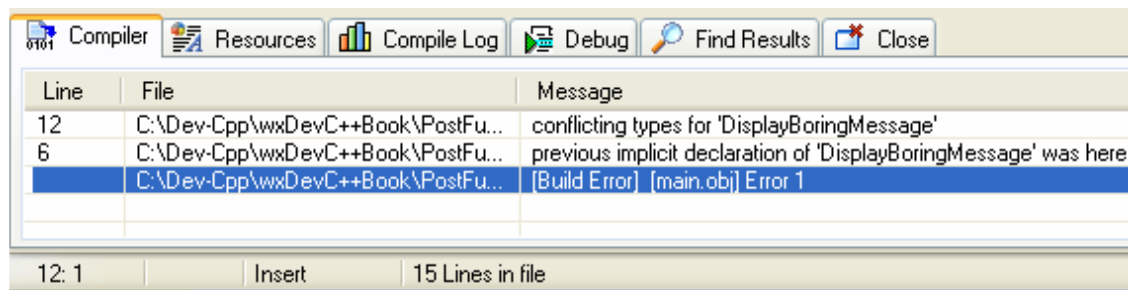


Figure 3.4 – Error message.

Get used to such messages you will see them a lot as you progress. But what was wrong? Let us try to work it out from the message. The first line says 'conflicting types for DisplayBoringMessage', that's the name of our function so what is wrong with it. Well the first error has come in line 12 of our source code. The next error message tells us that in line 6 there was a 'previous implicit declaration of 'DisplayBoringMessage''. Why is this? Because before you use a function C needs to know about it.

We can get around it by 'Declaring' the function that is by telling the compiler what the function name is, what the return type is and what parameters it takes. Try adding the following line into the source code in line 3.

```
void DisplayBoringMessage();
```

Since this is a declaration it ends with a semi-colon. Press <F9> to compile and run. You should see that the program runs just fine now. The line you added is called the ‘Declaration’ of the function and the code in lines 11-14 is called the ‘Definition’ of the function. When you include header files you are actually including the declaration of function like `printf` so that you can use them.

Before we move on to look at some common and useful functions that are included in the C standard library we will look at functions that can take a variable number of arguments.

This type of function is defined by using ‘...’ to say the function accepts a variable number of arguments. The function must contain at least one known parameter before the variable list of arguments. The reason for the known parameter is discussed after the example. The most common function to use this format is the `printf` function. The prototype of this style of function is

```
int MyFunction(int AnInteger, float AFloat = 5.4, ...);
```

So far so good, but how do we know how many extra arguments the user has passed in and how do we access them? For a start we need to include `<stdarg.h>` which defines various macros we need to use. Let’s look at a sample program first that uses a function with variable parameters then we will break it down.

Carry out the usual project creation process calling the new project VariableFunction.

Create a new folder called VariableFunction and save the project there.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <stdarg.h>
4
5  int AutoSumFunction (int NumberOfArguments, ...);
6
7  int main(int argc, char *argv[])
8  {
9      int Total = 0;
10     Total = AutoSumFunction (6, 12,54,24,75,45,23);
11     printf("The total is %d\n", Total);
12     system("PAUSE");
13     return 0;
14 }
15
16 int AutoSumFunction (int NumberOfArguments, ...)
17 {
18     int Sum = 0, CurrentValue = 0;
19     va_list ArgumentPointer;
20
21     /*We need to use va_start to get the start of the
22     argument list and store this in the argument pointer.
23     The macro va_start does this*/
24     va_start(ArgumentPointer, NumberOfArguments);
25
```

```

26     /*Now we will create a loop that counts down the number
27     of arguments.*/
28     for( ;NumberOfArguments;NumberOfArguments--)
29     {
30         /*We get the next value from the list of arguments,
31         we need to tell va_arg what type of value we expect*/
32         CurrentValue = va_arg(ArgumentPointer,int);
33         Sum = Sum + CurrentValue;
34     }
35
36     /*We must call va_end otherwise the program could crash*/
37     va_end(ArgumentPointer);
38
39     /*Return the total*/
40     return Sum;
41 }

```

Press <F9> to compile and hopefully you should see the output “The total is 233”. So let us breakdown this example to see what happens.

Line 3 - We include the header file <stdarg.h> this gives us access to `va_list`, `va_start`, `va_arg` and `va_end`.

Line 5 - We create the prototype of our `AutoSumFunction`. It is defined as returning an `int` value which is the sum of all its arguments. It takes one integer value which is the number of arguments we are going to pass in. Then it takes any number of other arguments.

Line 10 - We use the function passing in a 6 to say that we are going to give it six arguments. Then we pass in six integer values.

Line 19 - We use the data type `va_list` to create a pointer called `ArgumentPointer`. This is a pointer that we will use to access the list of arguments. We will learn more about pointers in the section ‘*Pointers*’.

Line 24 - We call the macro `va_start` passing it the pointer `ArgumentPointer` and the name of the first variable `NumberOfArguments`. This initialises our pointer to point to the first argument after `NumberOfArguments`.

Line 28 - We create a `for` loop which we will use to access the list of arguments. We don’t fill the first part of the `for` loop since `NumberOfArguments` already has a value. If you remember the second part of the `for` loop checks for a true/false value. We use `NumberOfArguments` here, which means if `NumberOfArguments` equals 0 then stop looping. The last part of the `for` loop decreases the value of `NumberOfArguments` by 1 each time.

Line 32 - We assign the return value from `va_arg` to `CurrentValue`. The parameters of `va_arg` are the `ArgumentPointer` pointer and the data type we expect the next argument to be.

Line 37 - We call `va_end` passing in the `ArgumentPointer`. Failing to do this is likely to lead to a program crash.

So we have seen how to access the variable list of arguments, but what about the other question, how do we know how many arguments there are? The answer unfortunately is we don't. That is why we need the first argument. To experiment with the truth of this

Alter line 10 to

```
Total = AutoSumFunction (6, 12,54,45,23);
```

Then run the program.

You should find the result vastly different to the sum of 12,54,45 & 23. You might also like to experiment with adding more arguments while leaving the first one as 6.

WARNING: Line 28 is somewhat dangerous we use the check `NumberOfArguments`. This will stop the loop when `NumberOfArguments` is equal to 0. But what if someone changed line 10 to `Total = AutoSumFunction (-6, 12,54,45,23);`. Then `NumberOfArguments` would never equal zero.

We will learn more about functions in the next two sections, '*Input/Output and other useful functions*' and '*Pointers*'.

Input/Output and other useful functions

Earlier I said that C contains no keywords dealing with output to the screen and that functions were used instead. We have so far seen two functions dealing with input and output, `printf` and `scanf`. The good news is if you learn about these two the rest of the input/output functions are fairly easy to learn.

Besides the input/output functions, we shall skim the wealth of functions C has available for the programmer.

Input/Output to the screen

Output

We shall start by looking at output, after all it is much more exciting to create a program that visibly does something, than one that you type a lot into and nothing seems to happen.

C has three output functions `putchar`, `puts`, `printf`. The first two are simple functions, `putchar` takes a `char` argument and prints it to the screen. The second takes

a string constant and prints it to the screen. The final one `printf` is far more complex and we will explore it in more depth.

The prototype of the `printf` function is:

```
int printf(const char* string, ...);
```

The return value from `printf` is either the number of characters it has written to the screen or a negative value if there has been an error. The first argument to `printf` is the string you wish to display in the console. For example:

```
printf("C Programming is amazing");
```

This would result in the line 'C Programming is amazing' being displayed in the console. But what about the second argument that string of dots? You may remember from our discussion of functions this represents a variable list of arguments. We have already used this feature in our sample programs. Within the string passed to `printf` as the first argument we can embed format specifiers at the points where we want to insert values. The format specifiers start with a '%' and these are also used in the function `scanf`. Every time the function finds a '%' sign embedded in your sting it looks for a matching argument and depending on the code after the '%' sign outputs the argument in this format. For example:

```
printf("I am %d today and my name is %s",200,"Rip Van Winkle\n");
```

In this example `printf` would output 'I am ' then it would meet the '%' sign it looks to see what follows in this case a 'd'. It now looks for the first variable argument which is '200', since the format code was 'd' it outputs this as an integer. Next it continues to output 'today and my name is ' again it comes to a '%' sign again it looks for the format sign in this case 's' so it formats the second argument as a string. A list of all the format specifiers follows.

Code	printf	scanf
%a	Hex output in form 0xh.hhhhp+d (C99)	Input a floating point (C99)
%A	Hex output in form 0xh.hhhhP+d (C99)	
%c	Output a character	Input a character
%d	Output signed decimal integer	Input a decimal integer
%e	Output in scientific notation lowercase e	Input a floating point
%E	Output in scientific notation uppercase E	
%f	Output decimal floating point	Input a floating point
%g	Output in the shorter of %e or %f	Input a floating point
%G	Output in the shorter of %E or %f	
%i	Output signed decimal integer	Input a decimal, octal or hexadecimal integer
%n	Stores number of characters written in an	Stores number of characters read in

	integer pointer	an integer pointer
%o	Output unsigned octal	Input an octal
%p	Displays pointer	Input a pointer
%s	Output character string	Input a string
%u	Output unsigned decimal integer	Inputs an unsigned decimal integer
%x	Output unsigned hexadecimal (lowercase)	Input a hexadecimal
%X	Output unsigned hexadecimal (uppercase)	
%%	Outputs '%' sign	Inputs a '%' sign
%[]		Reads a set of characters

The format specifiers in `printf` can also be modified in various ways. These are:

- **Minimum field width –** Placing an integer between the '%' sign and the format code ensures that the output is padded with spaces until it reaches the specified length. If it is longer than the specified length it is output in full.
- **Output precision -** The field width modifier can be followed by a decimal point and another integer. For floating point values this specifies the number of decimal places displayed. For strings this specifies the maximum number of characters displayed. For integers it specifies how many digits are displayed leading zero are used to pack the difference.
- **Output justification -** By default the output is right justified, by adding a minus sign after the '%' sign you can alter the output to be left justified.
- ***long* and *short* data type modification -** The modifiers 'l' for long and 'h' for short can be added to the format specifiers d,i,o,u,x and n to change them to long or short data types. The modifier 'L' can similarly be used with format specifiers e,f, and g to specify a long double.
- **Added decimal point -** Using a '#' sign after the '%' sign before g,G,f,e or E specifiers causes a decimal point to be displayed.
- **Added hexadecimal sign -** Using a '#' sign after the '%' sign before x or X specifiers causes a 0x prefix to be displayed.
- **Variable output precision -** Instead of using the format "%5.4f" it is possible to use "%*.*f". For each '*'

`printf` will look for the value to replace it from the list of variable arguments.

<p>NOTE: Remember when we discussed variable argument functions we said that <code>va_arg</code> needed to be told what data type was expected. That is the reason for the different format specifiers. Each time <code>printf</code> finds a ‘%’ in the string it looks for the next variable argument and returns it as the type listed in the above table. If your output from a <code>printf</code> function looks strange or gives wrong results make sure you are using the correct format specifiers. Also make sure you have the same number of arguments as format specifiers.</p>
--

Input

To match the three output functions C provide three input functions `getchar`, `gets` and `scanf`. Once again the first two are simple functions, `getchar` returns a `char` from the input buffer, `gets` takes a string array (there is more about arrays in the sections, ‘Pointer’, ‘Memory allocation’, ‘Arrays’ and ‘Strings’). The final function `scanf` is like its sibling `printf` a far more complex function that we will explore.

The prototype of the `scanf` function is:

```
int scanf(const char* string, ...);
```

The return value from `scanf` is the number of items that have been assigned a value or in the event of an error it returns EOF. EOF is a constant whose value can change from one compiler to another so just use the constant name EOF.

The first argument to the `scanf` function is a string of format specifiers, for each format specifier you need to include an argument which is a place for `scanf` to store the values it reads in from the keyboard. It works in a very similar way to `printf`, the main different is that the variable arguments all need to be addresses to variables. We will discuss addresses under the topic pointers, but for now just accept that this means putting a ‘&’ sign in front of each of the arguments. For example:

```
scanf("%d%f%s",&Int,&Float,&String);
```

In the above line the `scanf` would read the first input from the keyboard and store it in the variable ‘Int’, the second input would be stored in ‘Float’ and the final input in ‘String’.

There are a few hazards with `scanf`. The first is that in some implementations it buffers the lines. This means that if you use `scanf` to read single characters it won’t do anything

until you press [Enter]. Secondly when reading strings it reads in everything until the first white space character which is either a space, tab, vertical tab, formfeed or newline. For example `scanf` would read the line “Welcome friends and Romans” as “Welcome”. To read in a whole line including white spaces you need to use `gets`.

The scan set ‘`%[]`’ format specifier tells `scanf` to read in only those characters contained in the set. For example:

```
scanf( "[%GgeRhDW]", &String );
```

In the above example `scanf` will store the users input in the variable `String`. Any of the characters `G,g,e,R,h,d,W` will be stored. As soon as `scanf` encounters a character that is not in this range it will stop reading. The scan set is also case sensitive so `G` is not the same as `g`.

Like `printf` `scanf` supports various format modifiers these are as follows:

- Maximum field width - Inserting a number between ‘`%`’ and the format code will cause only that number of characters to be read.
- *Long* and *short* data type modifier - These are the same as `printf` `l`, `h` & `L`. These alter the length of the data type.
- Discard field - Inserting a ‘`*`’ between the ‘`%`’ and the format code will cause the input to be read but not assigned to any variable.

Since all this may be a lot to take in let’s create a sample program which makes use of these features. Take your usual steps to create a new C project.

Call the project ‘InputOutput’.
Save it in a new folder called ‘InputOutput’.
Alter the generated code to match the following

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main(int argc, char *argv[])
5  {
6      char FirstNameString[200];
7      char LastNameString[200];
8      int AgeInteger, Count;
9      float MyAgeFloat;
10     printf( "Hello what is your first name\n");
11     scanf( "%s",&FirstNameString);
12     printf( "\nHello %s, what is your last name?\n", FirstNameString);
13     scanf( "%s",&LastNameString);
14     printf( "\nSo you are %s %s\n", FirstNameString, LastNameString);
15     printf( "How old are you %s\n", FirstNameString);
16     scanf( "%hd", &AgeInteger);
17     MyAgeFloat = AgeInteger + 0.2;
```

```

18     printf( "\nThat's funny I am %*.2f\n",4,2,MyAgeFloat );
19     printf( "\nThis is right justified");
20     printf( "\n%23s\n",FirstNameString);
21     printf( "\nThis is left justified");
22     printf( "\n%-23s\n\n",LastNameString);
23     printf( "%s\n contains ",FirstNameString,&Count);
24     printf( "%d characters\n\n",Count);
25     system( "PAUSE");
26     return 0;
27 }

```

Press <F9> to compile and run.

The output should resemble this

```

C:\Dev-Cpp\wxDevC++Book\Src\InputOutput\InputOutput.exe
Hello what is your first name
Frank
Hello Frank, what is your last name?
Enstein
So you are Frank Enstein
How old are you Frank
634
That's funny I am 634.20
This is right justified
      Frank
This is left justified
Enstein
Frank contains 5 characters
Press any key to continue . . . _

```

Figure 3.5 – Output from InputOutput program

Most of the program should be self explanatory lines of note are:

Line 18 where the %*.2f format specifier is used. The next two arguments specify the total number of digits to display and the number of digits that appear after the decimal point.

Line 20 specifies that the output should be at least 23 characters wide. This causes the text to be right justified.

Line 22 specifies that the output should be 23 characters wide the ‘-‘ sign causes it to be left justified.

Line 23 outputs the users first name then stores the number of characters output into the variable Count. This is used in the next line to say how long the user’s first name is.

Next we shall look at the functions C provides to read and write files.

Input/Output to Files

The design of C is influenced by Unix which has the philosophy that everything is a file. This shows through in the design of the input/output functions. As a result the functions to read and write files are very similar to those you have already met. Actually the output console and input keyboard are also treated as files. These have the predefined names `STDOUT` and `STDIN`. The difference is that `printf`, `scanf` and the like use them transparently.

In order to read or write files you need to open them first and close them when you have finished with them. The header file you need to include is `<stdio.h>` this allows you to use the `FILE` type as a pointer to files.

To open we need to use the function `fopen`, the prototype of this function is defined in the following way.

```
FILE * fopen(const char * filename, const char * mode);
```

If the function succeeds the return type is a pointer to a valid `FILE` structure. If it doesn't succeed then it returns a `NULL` pointer. (If this makes as much sense to you as a speech in Klingon then look ahead to the sections on '*Pointers*' and '*Structures*'). The first argument is a string constant which contains a valid filename or file path. Examples of these are "Readme.txt" or "C:\\Windows\\Readme.txt". The second argument is a flag indicating what mode to open the file. The various options are contained in the following table.

Flag	Meaning
a	Open or create a text file and append to the end of it
r	Open a text file to read from it
w	Create a text file to write to it
ab	Open or create a binary file and append to the end of it
rb	Open a binary file to read from it
wb	Create a binary file to write to it
a+	Append or create a text file as read/write
r+	Open a text file to read/write from it
w+	Create a text file to read/write to it
a+b	Append or create a binary file as read/write
r+b	Open a binary file to read/write from it
w+b	Create a binary file to read/write to it

When you have finished using a file you then need to close it. In order to do this you need to use the function `fclose`. The prototype of `fclose` is declared this way.

```
int fclose(FILE * FilePointer);
```

If successful the `fclose` function returns 0 otherwise it returns EOF.

Output

The file output functions are very closely related to console output functions. They are so similar that they have the same names just with an `f` appended to the front of them, so we get `fputc()`, `fputs()` and `fprintf()`. The other major difference is that they all require a `FILE` pointer to a valid file. `fprintf()` is the only function we will discuss here. Its prototype is declared as:

```
int fprintf(FILE * filepointer, const char * string,...);
```

You may notice that other than the addition of the `FILE` pointer, it looks exactly like the prototype of the `printf()` function earlier. In fact it also works in exactly the same way as the sample program that follows the next section will demonstrate.

Input

Like the functions described in the previous section the file input functions are just the same as the console input functions just with an `f` appended to the front and an extra argument which is a valid `FILE` pointer. So we get `fgetc()`, `fgets()` and `fscanf()` once again it is `fscanf()` that we will discuss. The declaration of the prototype is as follows:

```
int fscanf(FILE * filepointer, const char * string, ...);
```

So let us look at an example of using these two function `fprintf` and `fscanf`. As usual

Start up a new C console project.
Call it SingleFile.
Save it in its own folder called SingleFile.
Then alter the generated source code to match the following.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main(int argc, char *argv[])
5  {
6      char Name[100];
7      int Age;
8      char NameFromFile[100];
9      int AgeFromFile;
10
11     FILE * OurFile;
12
13     /*Demonstrate that we can read and write to and from the console
14     using fprintf and fscanf with the pointers stdout and stdin*/
15     fprintf(stdout, "Please enter your first name and your age\n");
16     fscanf(stdin, "%s%d", Name, &Age);
17
18     /*Try to open a file to write to, check it is valid*/
```



```

19     if((OurFile = fopen("Test.txt" , "w")) == NULL)
20     {
21         /*If the file pointer is NULL complain and exit*/
22         fprintf(stdout,"Sorry but I can't open a file to write to\n\n");
23         exit(1);
24     }
25
26     fprintf(stdout,"\nWriting your information to file\n");
27     /*Use the OurFile pointer to write information to the file*/
28     fprintf(OurFile, "%s %d", Name, Age);
29     /*Close the file now we have finished with it*/
30     fclose(OurFile);
31
32     /*Try to open a file to read from, check it is valid*/
33     if((OurFile = fopen("Test.txt", "r")) == NULL)
34     {
35         /*If the file pointer is NULL complain and exit*/
36         fprintf(stdout,"Sorry but I can't open a file to read from\n\n");
37         exit(1);
38     }
39
40     fprintf(stdout,"Reading your information from file\n");
41     /*Use the OurFile pointer to read information from a file*/
42     fscanf(OurFile,"%s%d", NameFromFile, &AgeFromFile);
43     fclose(OurFile);
44
45     fprintf(stdout,"\nYour name is %s, ", NameFromFile);
46     fprintf(stdout,"and your age is %d\n", AgeFromFile);
47
48     system("PAUSE");
49     return 0;
50 }

```

Compile and run this program.

Some of this program should be fairly obvious from the program we used to demonstrate using printf and scanf. But I'll discuss some of the lines that differ.

Line 15 - The effect of this line is the same as using printf but instead we use fprintf and the pointer called stdout. This pointer as we discussed earlier writes directly to the console.

Line 16 - This line does the same as line 15 but used fscanf along with the pointer stdin to replace scanf.

Line 19 - We use the function fopen to attempt to open a file called Test.txt from writing to. We test the value of the returned pointer against the value NULL. If they match there has been an error while opening the file and we cannot use it.

Line 23 - If the file pointer is NULL we call the function exit with the value 1. This ends the program at this point and signals that the program has halted in an error state.

Line 28 - We write the values captured earlier to the file.

Line 30 - We close the file as soon as we have finished using it.

Line 42 - We read the values back in from the file, to prove that these are not just stored in memory we use different variables.

Line 43 - Once again we close our file as soon as we are finished with it.

While `fscanf` and `fprintf` are very powerful and fairly easy to use they are not the most efficient since they involve converting from binary to characters. For large amounts of data reading and writing there are other better options these are mentioned briefly in the next section but are beyond the scope of this book.

Other file handling functions

This is only the tip of the iceberg when it comes to file handling routines. If you are interested (and I know you are) then consider it a matter of homework to find out about `fseek`, `ftell`, `feof`, `ferror`, `rewind`, `remove`, `fflush`, `fread` and `fwrite`. These functions expand the file handling capabilities of C far beyond simple reading and writing of text files.

Some other useful functions

The following table lists some of the other useful functions that C provides.

Various functions – Contained in `<stdlib.h>`

<code>double atof(const char* str)</code>	Converts the string <code>str</code> into a double which it returns.
<code>int atoi(const char* str)</code>	As above but converts to an integer value.
<code>long int atoll(const char* str)</code>	As above but for a long integer value.
<code>int rand(void)</code>	Returns a random number between zero and <code>RAND_MAX</code> (this is at least 32,767).
<code>void srand(unsigned int seed)</code>	Sets a start point for the <code>rand</code> function.

There are many more functions contained in a range of header files that we haven't even discussed. It is well worth exploring what is available as there is little more annoying than struggling to write a function only to find that it has already been provided for you.

Pointers

So far we have mentioned pointers a few times indeed it is nearly impossible to talk about C at any length without mentioning pointers. They are one of the greatest strengths of C. Unfortunately they are also one of the greatest dangers in C programming. So what is a pointer?

The memory in a computer is like one great big road. All along the road are houses. Each house contains a piece of data. So how do we locate a piece of data. Well each house has

an address so we can use that address to locate the data. For some important buildings there may be a sign pointing to that building with a name on it. A pointer in C is exactly the same. The sign is a variable and the value it holds is the address within the computers memory of the start of the piece of data it points to.

To define a variable as a pointer we use a '*' after its data type and before the variable name. For example:

```
int * MyIntPtreter;
```

But how do we get the address of a piece of data to point to? Using the '&' operator before a variable name returns the address of that variables data. We saw an example of using this in the function `scanf`.

However most times you don't want to know where a piece of data is stored rather you are more interested in knowing what is stored there. To retrieve the value stored at the address the pointer references you can use the '*' operator. For example:

```
int AnInteger = *MyIntPtreter;
```

This assigns the value referenced by `MyIntPtreter` to the variable `AnInteger`.

WARNING: Like any variable in C when you define a pointer as above it points to a random value therefore it is good practice to assign it a value when you define it. If you have a value to point to then you can define it like this

```
int * MyIntPtreter = &AnInteger;
```

Otherwise you can assign it the value `NULL`.

```
int * MyIntPtreter = NULL;
```

`NULL` is used because it is guaranteed to point to nothing. Most functions that return a pointer to a section of memory will return `NULL` if there has been an error. In this case it is necessary to check what the return value is before using the returned pointer.

Uninitialised pointers are a common cause of errors in C programming. The best you can hope for are unexpected outputs. The worst is programs that crash or on older operating systems computers that crash. The reason for this is that you are operating on memory that belongs to another part of the program or to another program. Such problems can be very tricky to track down.

As we have already seen we can use pointers to point to variables we have already declared. We can also use pointers to allocate fresh memory. As we shall see in the next section.

To demonstrate pointers and the different operators they use let us create another example program. As usual

- Create a new C project.
- Name the project 'Pointy'.
- Save it in a new directory called 'Pointy'.
- Change the generated code to match the following.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main(int argc, char *argv[])
5  {
6      int Number1 = 5, Number2 = 12, Total = 0;
7      int * MyIntPtr;
8
9      /*Use MyIntPtr to reference Total. We need to use '&'*/
10     MyIntPtr = &Total;
11     /*Use MyIntPtr to display value and location of variable Total*/
12     printf("The value of Total = %d.\n", *MyIntPtr);
13     printf("The Location of Total is %d\n", MyIntPtr);
14     /*Use MyIntPtr to alter the value of Total*/
15     *MyIntPtr = Number1 * Number2;
16     /*Display the location and value of Total
17     displaying that MyIntPtr changed it*/
18     printf("The value of Total = %d.\n", Total);
19     printf("The location of Total is %d\n", &Total);
20
21     system("PAUSE");
22     return 0;
23 }
```

This program demonstrates the use of pointers to manipulate the values held in a memory location. By now the lines up to line 9 should make sense to you. Let us consider some of the other lines.

Line 10 - uses the '&' address operator to assign the address of the variable Total to the pointer MyIntPtr.

Line 12 - uses the '*' operator to print out the value held in the memory location pointer to by MyIntPtr.

Line 13 - then prints out the value of MyIntPtr which is the memory location of Total.

Line 15 - assigns the value of Number1 * Number2 to the memory location pointed to by MyIntPtr.

Line 18 - out the value of Total showing that it has been altered by means of the MyIntPtr.

We will learn more about pointers in the following sections 'Memory allocation', 'Arrays' and 'Strings'.

Memory allocation

Within C programming a distinction is made as to where the memory you use is allocated from. There are two terms used the *stack* and the *heap*. It is not vital to know about these but you will come across the terms every now and then so it is useful to understand the difference.

As your program runs it allocates memory from the stack. As each function is called it is placed on the memory stack, any variables declared within the function are also placed on the stack. As the function ends it is removed from the stack and any variables local to function are destroyed (as you may remember from the discussion on scope earlier).

Sometimes though you want to store data within a function yet use it later. If this information was stored on the stack it would be destroyed when the function returns. To avoid this you can allocate memory on what is called the heap. You are responsible for deciding when memory you require from the heap is created and destroyed. C provides a library of functions dedicated to memory allocation and destruction. We shall look at these now.

The header file we need to include to use these functions is `<stdlib.h>`. There are four functions that can be used `malloc`, `calloc`, `realloc` and `free`. We shall consider each of these.

`malloc` The function prototype is

```
void* malloc(size_t SizeInBytes);
```

To use the function it is necessary to specify the amount of memory you want to ask for, `malloc` then returns a pointer which references the location of this memory. This pointer is a void pointer which you need to cast to the correct type for your use. If the function is unable to allocate the memory it will return a NULL pointer. For Example:

```
int * MyIntPtr = NULL;
MyIntPtr = (int*) malloc(sizeof(int));
If(MyIntPtr)
...
```

First we declare `MyIntPtr` to be a pointer to type `int`. We assign it the value `NULL`. Next we use `malloc` and ask for a piece of memory the size

of an `int`. We cast the returned pointer from `malloc` from type `void` to type `int`. Then assign this to `MyIntPtr`. Finally we check the `MyIntPtr` does not point to `NULL` before using it.

`calloc` The memory returned by `malloc` contains random values. `calloc` initializes all the bits in the returned memory to zero. The prototype for `calloc` differs from `malloc` and is declared as

```
void * calloc (size_t Number, size_t Size);
```

To use `calloc` we need to specify the number of units we want and the size of these units. For example:

```
int * MyIntPtr = NULL;
MyIntPtr = (int*) calloc(1, sizeof(int));
If(MyIntPtr)
...

```

This example is the same as the one for `malloc` the difference is that we ask for one unit of size `int`. The returned memory will be allocated to zero.

`realloc` Sometimes you may find that you haven't allocated enough memory and need to enlarge your allocation that is when you need `realloc`. The prototype for `realloc` is

```
void * realloc(void * ExistingPointer, size_t
SizeInBytes);
```

To use `realloc` we need to use our pointer to the existing block of memory. Followed by the size we need the new block to be. `realloc` returns a pointer to the new memory block. Again this can be `NULL` if there has been an error.

`free` Once you are done with the memory you have created with the above functions you need to free the memory allocated. You can do this with the `free` function. The prototype for `free` is declared as

```
void free(void * ptr);
```

You call `free` with the pointer that you have received back from one of the above operations.

<p>WARNING: It is vital that you free all memory that you have previously allocated. Otherwise when your program ends this memory can still be allocated. This section of memory is then never freed for other programs to use. This</p>

is called a *memory leak*. In low memory situations programs with memory leaks can cause operating systems to crash.

Arrays

An array is a block of memory consisting of a number of the same type of data units. There are many instances in which using arrays makes sense. For example you may wish to store the ages of your family members. You could do this by creating a new variable for each member. E.g.

```
int Member1 = 34, Member2 = 54, Member3 = 23;
```

To print out a list of ages you could do the following

```
printf( "Member 1 = %d, Member 2 = %d, Member 3 = %d",  
Member1, Member2, Member3 );
```

This is fine for a few family members. But what if you have a huge family? That `printf` statement is going to get unwieldy. The answer is an array.

An array is defined in the same manner as a variable the difference is that after the variable name you enclose the number of units you need in square `[]` brackets. For example:

```
int FamilyMembers [10];
```

This would give you an array of 10 integers named `FamilyMembers`. An array is not much use unless you can fill it with values and retrieve them. So how do we fill it? The first way is by filling it when we declare it to do this we put an equals `=` sign after the square brackets and then list the values between `{ }` brackets. For example:

```
int FamilyMembers[3] = {24, 54, 63};
```

This is called array initialization. It is also possible when initializing arrays at the same time as declaring them as above to miss out the array size in the square brackets. The compiler will automatically work out the size of the array for you for huge arrays and strings this can be a blessing.

Alternatively you can fill each section of the array by referring to it by its index number. One important point is that in C arrays start at index 0. So in the previous example `FamilyMembers` starts at index 0 and ends at index 2. So we can refer to them this way:

```
FamilyMembers[1] = 45;  
printf( "Uncle Fred is %d years old", FamilyMembers[1] );
```

What is not immediately apparent is that all arrays in C involve using pointers, this is because the array name is just a pointer to the memory location of the first element of an array. So the above could be written as:

```
*(FamilyMembers + 1) = 45;
printf("Uncle Fred is %d years old",*(FamilyMembers +
1));
```

So let us look at an example programme that demonstrates using arrays and pointers. As per usual

Create a new C project.
Call it 'TodaysArray'.
Save it in a folder called 'TodaysArray'.
Alter the code to match the following.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  /*Create a define for our array size*/
5  #define DAYS_IN_A_WEEK 7
6
7  int main(int argc, char *argv[])
8  {
9      /*Create array and fill it*/
10     int TodaysLuckyNumber[DAYS_IN_A_WEEK] = {35,64,23,86,23,12,43};
11     int ArrayIndex = 0;
12
13     /*Loop through array and print out the days lucky number*/
14     for(;ArrayIndex < DAYS_IN_A_WEEK; ArrayIndex++)
15         printf("For day number %d the lucky number is %d\n",
16             ArrayIndex + 1, TodaysLuckyNumber[ArrayIndex]);
17
18     /*Reset the index*/
19     ArrayIndex = 0;
20     /*Print a new line*/
21     printf("\n");
22
23     /*Now do it all again using pointers*/
24     for(;ArrayIndex < DAYS_IN_A_WEEK; ArrayIndex++)
25         printf("For day number %d the lucky number is %d\n",
26             ArrayIndex + 1, *(TodaysLuckyNumber + ArrayIndex));
27
28     system("PAUSE");
29     return 0;
30 }
```

As usual let us break down the lines of interest to us.

Line 5 - We define a constant value to use in creating the array and for use in the loop that indexes the various elements. The beauty of this is that if you need to resize the array you can alter the value of the constant. The constant also helps the program to be self documenting.

Line 10 - We create an integer array called `TodaysLuckyNumber`. We fill the contents of the array at the same time as we create it.

Line 14 - We create a `for` loop the first part is empty since we have already initialised `ArrayIndex` to 0. We check `ArrayIndex` is lower than `DAYS_IN_A_WEEK` since if you remember the array starts at 0 and ends at `size - 1`.

Line 16 - We print out the value of the index plus one and the value contained in `TodaysLuckyNumber[index]`.

Line 26 - We repeat the actions of line 16 the difference is in the second part where we use the variable name as a pointer. We use `*` to obtain the value from `TodaysLuckyNumber`. We add `ArrayIndex` to the value of `TodaysLuckyNumber` to move the pointer along. We need to include this in braces otherwise we end up adding the value of `ArrayIndex` to the contents of `TodaysLuckyNumber` (remember operator precedence).

It is possible to create 2, 3 or more dimensional arrays, but that is beyond the scope of this book. Again if you are interested there are many excellent resources on the Internet to help you. Our discussion of arrays leads us next to consider string handling in C.

Strings

String handling in C is considered by many to be one of its weak points. The main reason for this is that strings are implemented using arrays of type `char`. This is different to other programming languages which consider a string to be a data type in its own right. As a result strings have a tacked on feel about them and require functions to achieve basic operations.

We saw in the above section how to declare and initialize arrays at the same time we can do the same with strings. Remember that a string is just an array of characters. So we can initialize it like this:

```
char MyString[] = {'I', ' ', 'a', 'm', ' ', 'b', 'o', 'r', 'e', 'd', '\\0'};
```

Wow! That is so long winded. There must be an easier way to initialize strings. Luckily there is. The above example can be declared and initialized in the much more user friendly manner of:

```
char MyString[] = "I am bored";
```

Ah, but what a minute surely those two lines don't have the exact same meaning. The first has a `'\0'` thing at the end and the second one doesn't. The `'\0'` thing on the end is the null terminator. This tells functions that are working with the string where the string ends. It is missing in the second line because using the shorthand method of initialization automatically adds the null onto the end. So the second method is not just easier, it's quicker, smarter and washes the dishes.

So what can we do with strings? We have already seen that we can print them out and read them in. However with the use of various functions there are all sorts that we can do with them. We can compare strings, add strings to other strings, find out how long they are and much more.

The following table shows some of the various string and character operation functions plus the header files they are contained in.

String functions – Contained in <string.h>

<code>char * strcpy (char * str1, const char * str2)</code>	Copies <i>str2</i> into <i>str1</i> . <i>str2</i> must be null terminated
<code>char * strcat (char * str1, const char * str2)</code>	Adds <i>str2</i> to the end of <i>str1</i> and adds a null terminator, you need to make sure <i>str1</i> is long enough for this
<code>char * strchr (const char * str, int ch)</code>	Returns a <code>char</code> pointer to the first occurrence of the character <i>ch</i> in the string <i>str</i>
<code>int strcmp (const char * str1, const char * str2)</code>	Compares <i>str1</i> to <i>str2</i> . Returns a zero value if both strings are the same, a negative value if <i>str1</i> is less than <i>str2</i> and a positive value if <i>str1</i> is greater than <i>str2</i>
<code>size_t strlen (const char * str)</code>	Returns an integer containing the length of a null terminated string <i>str</i> . The terminator is not counted.
<code>char * strstr (const char * str1, const char * str2)</code>	Returns a <code>char</code> pointer to the first occurrence of the string <i>str2</i> in the string <i>str1</i>

Character functions – Contained in <ctype.h>

<code>int isalnum(int ch)</code>	Returns a non zero value if the character <i>ch</i> is a letter or number
<code>int isalpha(int ch)</code>	Returns a non zero value if the character <i>ch</i> is a letter
<code>int isspace(int ch)</code>	Returns a non zero value if the character <i>ch</i> is a white space character. E.g. tab, space, etc
<code>int isdigit(int ch)</code>	Returns a non zero value if the character <i>ch</i> is a number

<code>int isupper(int ch)</code>	Returns a non zero value if the character <i>ch</i> is an upper case letter
<code>int islower(int ch)</code>	Returns a non zero value if the character <i>ch</i> is a lower case letter
<code>int ispunct(int ch)</code>	Returns a non zero value if the character <i>ch</i> is a punctuation mark. E.g. ! or ?
<code>int tolower(int ch)</code>	Returns the lower case equivalent of the character <i>ch</i>
<code>int toupper(int ch)</code>	Returns the upper case equivalent of the character <i>ch</i>

We will look at a brief example of using a few of these functions. So fire up wxDev-C++ and do the following:

- Create a new C project
- Name the project 'MyStringThing'
- Create a new folder called 'MyStringThing'
- Save your project there
- Alter the generated source code to match the following

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <ctype.h>
4  #include <string.h>
5
6  int main(int argc, char *argv[])
7  {
8      /*Declare and initialise our variables*/
9      char FirstName[50] = "";
10     char SecondName[50] = "";
11     char FullName[110] = "";
12     int LoopIndex = 0;
13     int UpperCase = 0;
14     int LowerCase = 0;
15     int Punctuation = 0;
16
17     printf("Please enter your first name\n");
18     scanf("%s", FirstName);
19     printf("Please enter your family name\n");
20     scanf("%s", SecondName);
21
22     /*Copy FirstName into FullName*/
23     strcpy(FullName, FirstName);
24     /*Use strcat to add a space after first name*/
25     strcat(FullName, " ");
26     /*Use strcat to append the FamilyName*/
27     strcat(FullName, SecondName);
28

```

```

29     printf("\nYour full name is %s\n",FullName);
30     printf("Your first name is %d characters long\n",strlen(FirstName));
31
32     /*Loop through each letter in the string and keep a count of
33     upper and lower case letters plus punctuation marks*/
34     for(LoopIndex = 0; LoopIndex < strlen(FullName); LoopIndex++)
35     {
36         if(islower(FullName[LoopIndex]))
37             LowerCase++;
38         else if (isupper(FullName[LoopIndex]))
39             UpperCase++;
40         if(ispunct(FullName[LoopIndex]))
41             Punctuation++;
42     }
43
44     printf("\nYour full name contains:\n");
45     printf("%-3d Lower case letters\n",LowerCase);
46     printf("%-3d Upper case letters\n",UpperCase);
47     printf("%-3d Punctuation mark",Punctuation);
48
49     /*use the shorthand if ?: to add an s to the end of the previous
50     output if punctuation equals anything other than 1*/
51     printf("%c\n\n",Punctuation == 1?' ':'s');
52
53     system("PAUSE");
54     return 0;
55 }

```

Compile and run the program.

You should receive an output resembling the following

```

C:\Dev-Cpp\w\DevC++\Book\Src\Chapter3\MyStringThing\MyStringThing.exe
Please enter your first name
Carlos
Please enter your family name
Blake-Funtington

Your full name is Carlos Blake-Funtington
Your full name first name is 6 characters long

Your full name contains:
18 Lower case letters
3 Upper case letters
1 Punctuation mark

Press any key to continue . . .

```

Figure 3.6 – Output from the MyStringThing Program

Let us have our usual breakdown (of the program)

Line 3 - We include the new header file `<ctype.h>` this contains the character handling functions.

Line 4 - We include the header file `<string.h>` this contains the string handling functions.

- Line 23 - We use the function `strcpy` to copy the contents of the string `FirstName` into the string `FullName`.
- Line 25 - We use the function `strcat` to add the string constant " " to the end of the string `FullName` this gives us a space between the first name and the family name.
- Line 26 - We use the function `strcat` to add the contents of the string `FamilyName` to the end of the string `FullName`.
- Line 30 - We use the function `strlen` to output the length of the first name.
- Line 34 - We start a `for` loop to look at the string character by character.
- Line 36 - We use `islower` function to see if this character is lower case or not.
- Line 38 - If this character was not lower case we use the `isupper` function to see if it is uppercase or not.
- Line 40 - We use the `ispunct` function to see if this character is punctuation or not.
- Line 51 - We use the shorthand if operator `?:` to output an `s` after punctuation mark if we have higher or lower than 1 punctuation mark, this makes our program a little more professional.

Now for something a little different, creating our own data types. We consider creating datatypes that consist of several pieces of data in the next section Structures.

Structures

So far we have seen how to use arrays to group together items of data of the same kind. But what about grouping together items of different kinds? To do this C provides structures. There are many uses for structures. One of these is to overcome the limitation that functions can only return one piece of data. By making that piece of data a structure you can return many items at once. To create a structure we need to use the aptly named keyword `struct`.

struct

You have already come across a structure when we looked at input/output with files. The `FILE` data type is actually a structure. Besides the keyword `struct` we also need a name to refer to the structure by and data members. An example of a structure follows below.

```
struct CarDetails
{
    int Age;
```

```

    char Model[30];
    char Registration[10];
};

```

After the keyword `struct` comes the name of the structure then inside the ‘{ }’ braces are the structures *members*. If you are used to Pascal programming or databases then a structure corresponds to a record and the members to a field.

We can use the structure as a data type the only caveat is that we need to use the word `struct` in front of it. For example to use the above example we can write the following:

```

struct CarDetails MyCarDetails;

```

In the above example we create a new structure called `MyCarDetails`. But having created the new structure how do we access the members? This is done using the dot ‘.’ operator. For example:

```

MyCarDetails.Age = 10;
printf("My car is %d years old\n",MyCarDetails.Age);

```

So far so simple. However if the structure is referred to via a pointer the method of accessing its members alters. Now we need to use ‘->’ instead of ‘.’. For example:

```

struct CarDetails ACarDetails;
struct CarDetails * MyCarDetails = &ACarDetails;
MyCarDetails->Age = 10;
printf("My car is %d years old\n",MyCarDetails->Age);

```

It is also possible to use the dot ‘.’ operator with pointers using the following notation.

```

(*MyCarDetails).Age = 10;

```

Due to the order of precedence rules it is necessary to use the braces around `*MyCarDetails`.

Although the structure has become like user defined data type it is not quite the same since you have to use the keyword `struct` in front of the structure name when ever you wish to create a new instance of it. However by making use of another C keyword ‘`typedef`’ we can over come this restriction.

`typedef` is used to assign another name to a data type for instance

```

typedef int MyInt;

```

This allows you to use MyInt as a data type. We can combine this with structures in the following way.

```
typedef struct
{
    int Age;
    char Model[30];
    char Registration[10];
}CarDetails;

CarDetails MyCarDetails;

MyCarDetails.Age = 10;
```

This allows you to use the structure exactly as you would any other data type. Along with the structure C provides a similar option called the union.

union

The union looks exactly like a structure the difference is that each member of a union occupies the same area of memory therefore you can only use one member of a union at a time. Therefore while a structure is the size of all its members combined, a union is the size of its largest member. A union has several uses, one of which is to save memory when you only need to operate on one member at a time. Using a union is in all other respects like using a structure. An example follows:

```
typedef union
{
    int Age;
    char Model[30];
    char Registration[10];
}CarDetails;

CarDetails MyCarDetails;

MyCarDetails.Age = 10;
```

If we hadn't used typedef then we would have needed to place the name CarDetails after the keyword union and have used the keyword union before CarDetails MyCarDetails;.

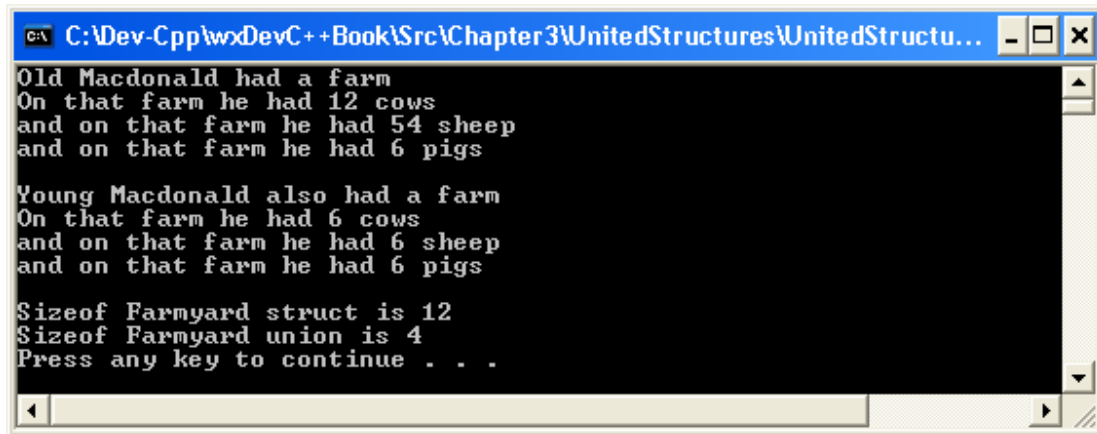
NOTE: This restriction is removed in C++ therefore you can do the above in C++ without resorting to using typedef and your structure will work like any other data type.

To give you some idea of structures and unions let us create a sample program. As usual

Create a new C project
Call it UnitedStructures
Save it in its own folder called 'UnitedStructures'
Alter the generated source code to match the following.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  typedef struct
5  {
6      int NumberOfCows;
7      int NumberOfSheep;
8      int NumberOfPigs;
9  }FarmYardStruct;
10
11 typedef union
12 {
13     int NumberOfCows;
14     int NumberOfSheep;
15     int NumberOfPigs;
16 }FarmYardUnion;
17
18 int main(int argc, char *argv[])
19 {
20     /*Create a structure*/
21     FarmYardStruct OldMacFarm;
22     /*Create a union*/
23     FarmYardUnion YoungMacFarm;
24
25     /*Fill the members of OldMacdonaldsFarm structure*/
26     OldMacFarm.NumberOfCows = 12;
27     OldMacFarm.NumberOfSheep = 54;
28     OldMacFarm.NumberOfPigs = 6;
29
30     /*Fill the members of YoungMacdonaldsFarm union*/
31     YoungMacFarm.NumberOfCows = 12;
32     YoungMacFarm.NumberOfSheep = 54;
33     YoungMacFarm.NumberOfPigs = 6;
34
35     /*Lets tell the world all about the Macdonalds*/
36     printf("Old Macdonald had a farm\n");
37     printf("On that farm he had %d cows\n",OldMacFarm.NumberOfCows);
38     printf("and on that farm he had %d sheep\n",OldMacFarm.NumberOfSheep);
39     printf("and on that farm he had %d pigs\n\n",OldMacFarm.NumberOfPigs);
40
41     printf("Young Macdonald also had a farm\n");
42     printf("On that farm he had %d cows\n",YoungMacFarm.NumberOfCows);
43     printf("and on that farm he had %d
44     sheep\n",YoungMacFarm.NumberOfSheep);
45     printf("and on that farm he had %d
46     pigs\n\n",YoungMacFarm.NumberOfPigs);
47
48     /*Print out the sizes of the structure compared to the union*/
49     printf("Sizeof Farmyard struct is %d\n",sizeof(FarmYardStruct));
50     printf("Sizeof Farmyard union is %d\n",sizeof(FarmYardUnion));
51
52     system("PAUSE");
53     return 0;
54 }
```


As usual compile and run the program. You should get something similar to the following output.



```
C:\Dev-Cpp\work\DevC++\Book\Src\Chapter 3\UnitedStructures\UnitedStructu...
Old Macdonald had a farm
On that farm he had 12 cows
and on that farm he had 54 sheep
and on that farm he had 6 pigs

Young Macdonald also had a farm
On that farm he had 6 cows
and on that farm he had 6 sheep
and on that farm he had 6 pigs

Sizeof Farmyard struct is 12
Sizeof Farmyard union is 4
Press any key to continue . . .
```

Figure 3.7 – Output from UnitedStructures program

So why is there a difference between Old Macdonald's farm and Young Macdonald's farm?

The difference is that the details of Young Macdonald's farm are held in a union. As the number of animals is written to each member it over writes the previous assigned value since all the members share the same memory space.

This is proved by the output from lines 47 & 48. The size of the FarmYardStruct is 12 or 3 times the size of an integer. Whereas the size of the FarmYardUnion is 4 the size of a single integer.

enum

While the enum does not strictly belong in a discussion about structures, it provides another way to create your own data types and is declared in a similar way. An example of an enum follows:

```
enum Monsters { Godzilla, KingKong, Predator, Alien, GeorgeClooney };
```

The above lists a set of integer constants. The first Godzilla is numbered 0 and each one after that is one number higher. The tag Monster can now be used as a data type, e.g.

```
enum Monsters MyMonster;
MyMonster = Godzilla;
```

It is also possible to initialise the values of any or all of the sets members. E.g.

```
enum Monsters {Godzilla = 5, KingKong, Predator = 12, Alien};
```

In this instance Godzilla has the value of 5, KingKong since it is next in sequence has the value of 6, Predator has the value of 12 and Alien has the value of 13.

Enumerations are of great use in `switch` statements where they can improve the human readability of the code. For example:

```
switch (MyMonster)
{
    case Godzilla:
        printf("Hey what a cool monster\n");
        break;
    case KingKong:
        printf("He's not so scary\n");
        break;
    case Predator:
        printf("Now you're talking\n");
        break;
    case Alien:
        printf("Now I am getting worried\n");
        break;
    case GeorgeClooney:
        printf("Agh, keep it away from me!\n");
        break;
    default:
        printf("No body is scared of the default monster\n");
}
```

That brings us to the end of our discussion of structures and almost to the end of our discussion on C programming, by now you are well and truly on your way to the ranks of C programmers.

Summary

This concludes our very brief tutorial in C programming. We have considered the basic data types, the most common keywords and some of the many functions available in the standard libraries. This by no means covers the whole of C programming and if you have come this far you may wish to go further. Part 4 of this book and particularly the Resources section points the way to a few of the many web based resources available for continuing your learning.

Now we shall continue with the enhancement to C, C++. This adds many features to C to make it more suitable for the object oriented programming paradigm. To learn more continue to the next chapter Basic C++ Programming.

Chapter 4 – Basic C++ Programming

Introduction

C++ began life as an enhancement to C. The name is derived from C's increment operator '++' showing that C++ is the next progression of C. It was designed by Bjarne Stroustrup while working at Bell Labs. Stroustrup wanted to make C more practical for programming large applications. He was influenced by the features found in other languages, particularly Object Orientation. (We will look at Object Orientation in the section 'Classes'). C++ improves upon C in two major ways. The first are the additions to the language. C++ now has 62 keywords the extra ones implement new features some of which have found their way back into C99. The second change is to the libraries. C++ introduces new Input/Output libraries, namespaces and most dramatically the STL (Standard Template Libraries).

C++ is like C a growing language and there are different standards which have been agreed upon and released. The first was in 1998 and the second in 2003 various features available in the later standard such as the STL are unavailable in the 1998 standard. At present a new standard C++0X is being drawn up.

C++ introduces the following features. Single line comments beginning with '//'. Name spaces to create encapsulation. Classes to enable object oriented programming. Memory allocation and retrieval with new/delete. Function and operator overloading. Exception handling, templates and RTTI (RunTime Type Identification). There are many other changes that we don't have room for.

One of the most touted benefits of C++ is its backward compatibility with C. This is true to a point. Most older C programs will compile without too much trouble with C++. One problem is C++'s new keywords which may have been used as variable names or the like in an old C program. Since the introduction of C99 the C language has also added new keywords which don't exist in C++ so that also breaks the compatibility.

We will continue now by looking at a basic C++ program that introduces some of the new features found in C++.

Break down of a simple example

Like we did before in our introduction to C programming we will create a small example in wxDev-C++ of a C++ program, compile it and see what it does. Then we will go on to break it down and examine it in detail.

So start up wxDevC++ and create a new project. Select a 'Console Application' and make sure the radio button for C++ Project is selected. Name the project 'SimpleC++'.

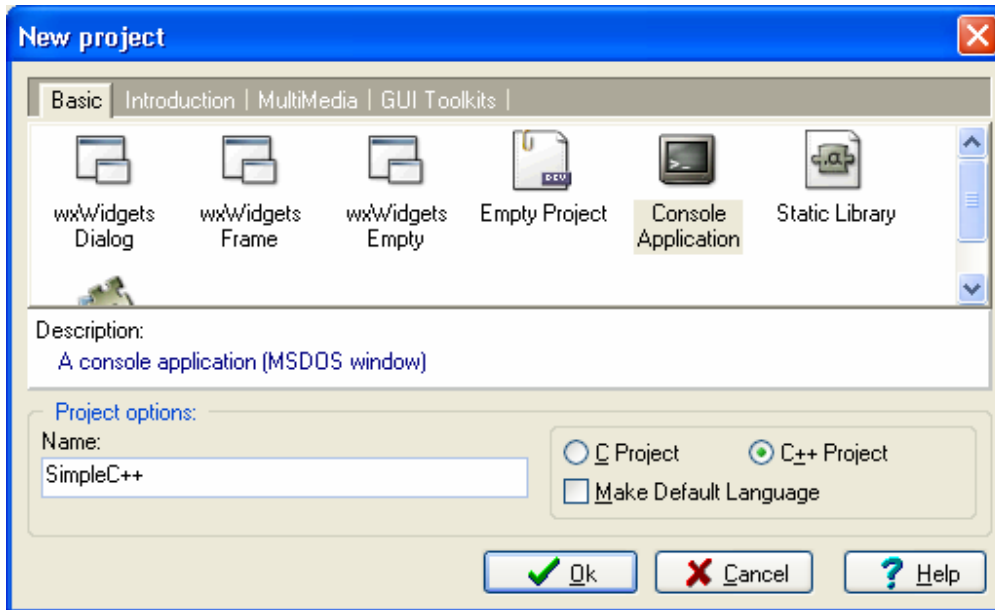


Figure 4.1 – Project settings for new SimpleC++ application.

In the next dialog create a new folder called 'SimpleC++' and save the project there. The IDE should now create the following basic source code for you.

```

1  #include <cstdlib>
2  #include <iostream>
3
4  using namespace std;
5
6  int main(int argc, char *argv[])
7  {
8      system("PAUSE");
9      return EXIT_SUCCESS;
10 }

```

Now change this to the following

```

1  #include <cstdlib>
2  #include <iostream>
3
4  using namespace std;
5
6  const int TOP_MAN = 2;
7  const int THE_PRISONER = 6;
8
9  int main(int argc, char *argv[])
10 {
11     cout << "Who are you?" << endl;
12     cout << "I am the new number " << TOP_MAN;
13     cout << ", You are number " << THE_PRISONER << endl << endl;
14     //Just a comment to show off C++'s single line comment style
15     for(int i = 0; i < THE_PRISONER; i++)
16     {
17         cout << "Thinking...\n" << endl;
18     }
19 }

```

```
20     cout << "I am not a number, I am a free man!" << endl;
21
22     system( "PAUSE" );
23     return EXIT_SUCCESS;
24 }
```

Once you have done this press <F9> to compile and run. The output should be exactly the same as for the ‘SimpleC’ program in the last chapter. Let us now break down the code to see the differences in the way that C++ achieves this compared to C.

Lines 1 and 2 should look fairly familiar they include the libraries we need to use. The first difference you may notice is that the included files don’t end in a ‘.h’. I am not quite sure why this is but I suspect that it is due to the fact that before the standards were drawn up several compilers created their own versions of libraries that later became standard. Thus there are deprecated headers called ‘iostream.h’ and so on.

Line 4 introduces two of C++’s new keywords `using namespace`. Namespaces provide a way to create functions with the same names and parameters and use them in the same program. <More here about namespaces>

Lines 6 and 7 show an alternative to preprocessor defines. These lines use the keyword `const` which means the value of the variable cannot change. Unlike defines constants have to have a type like `int` or `float`, etc.

Line 11 introduces us to the new C++ output function `cout`. `cout` uses `<<`, something about `std::cout` and `std::endl` more in input/output functions.

Line 14 comment

Line 15 declaration of `int` in the `for` loop not allowed in C.

Line 17 shows that escape characters such as ‘\n’ still work along with `endl`;

Basic C++

C++ (structs changed, `bool` data type only takes two values true and false)

Functions

C++ makes a few alterations to the good old function. Once you have got used to these you will wonder how we managed without them for so long.

Predefined values

Remember back in the last chapter we spoke of functions which took variable length arguments? Well C++ has added to this by allowing you to create functions that have default values pre-filled.

This is achieved by filling in the values in the function prototype. For example

```
int MyFunction(int AnInteger, float AFloat = 5.4, double ADouble = 43.7);
```

You don't need to define all the values, but once you have defined one you need to define all the others that follow it. Now you can use this function like `MyFunction(2);` or `MyFunction(2, 5.3, 64);` The first usage would automatically fill in the the values for `aFloat` and `ADouble` as `5.4` and `43.7` respectively.

Passing by reference

Before when we looked at function in C we mentioned using data types as arguments to a function. For example:

```
int MyExampleFunction( int AnInt, int AnotherInt );
```

The problem with using this type of function is that a copy of the variable you pass to the function is made in the computers memory. This takes time and memory. If you are repeatedly using the same function or using a large user defined data structure this can create an unacceptable level of overhead.

The answer to this problem in C is to use pointers. For example:

```
int MyExampleFunction( int * AnInt, int * AnotherInt );
```

However the problem with this is that pointers are inherently dangerous and their usage is recommended to be as limited as possible. C++ provides a new feature which gives you the best of both worlds. This is called passing by reference. To use this function you use the `'&'` operator before the argument names. For example:

```
int MyExampleFunction( int & AnInt, int & AnotherInt );
```

You would call this function just as you would call any other function. For example:

```
ReturnValue = MyExampleFunction( FirstInt, SecondInt);
```

Unlike using pointers you don't need to include a `'&'` before `FirstInt` and `SecondInt`. But the variables are acted on directly rather than on copies.

Inline

Sometimes you may have a function that is going to be called thousands of times in your program. This is going to create a bottleneck in the program. However the computer has various tricks up its sleeve to speed up execution of various parts of your program. Adding the keyword `inline` before the function indicates to the compiler that you want this function to be optimised. For example:

```
inline int MyExampleFunction( int AnInt, int AnotherInt );
```

The compiler then tries to do its best to comply. However there is no guarantee that the function will be optimised. The only guarantee is that the compiler will try.

Overloading

For me this is the best addition to functions. Sometimes you will write two or more functions that do almost exactly the same thing, but take different arguments. For an example consider the case where you have three functions which return the sum of their arguments. The first takes two ints, the second two floats and the third two doubles. The question is what to name these functions. In C you can't have two functions with the same name so you may end up with:

```
int SumInt(int AnInt, int AnotherInt);  
float SumFloat(float Afloat, float AnotherFloat);  
double SumDouble(double ADouble, double AnotherDouble);
```

How much simpler if you could simply call the function Sum and then pass in any of the three arguments. Well with C++ you can. For example:

```
int Sum(int AnInt, int AnotherInt);  
float Sum(float Afloat, float AnotherFloat);  
double Sum(double ADouble, double AnotherDouble);
```

This makes for much neater solutions (it also saves us lazy programmers from wearing our brains out thinking up new function names). There are not many rules for creating overloaded functions. The rules are:

1. The function must differ on more than just the return type. This is not allowed:

```
int Sum( int AnInt, int AnotherInt);  
float Sum ( int AnInt, int AnotherInt);
```

The reason for this is the compiler would not be able to tell which function you intended to call.

2. The arguments must not be ambiguous. This is not allowed:

```
int Sum( int AnInt, int AnotherInt);  
int Sum( int AntInt, int AnotherInt, int OneMoreInt = 40);
```

The reason is that the compiler would not know whether the line `ReturnValue = Sum(4, 6);` was supposed to call the first function with two arguments, or the second function with the third argument as the default value.

3. You have to send the author of this book \$20 every time you use an overloaded function. (Okay I lied on the last one, but feel free to do as you wish).

So as you can see there have been some great improvements in the area of functions.

Memory allocation

I'm sure you will remember back in the last chapter we touched on memory allocation. To achieve this in C we needed to make use of various functions `alloc`, `malloc`, etc. C++ takes a different approach and provides two new keywords `new` and `delete` to enable you to manage allocating and de-allocating memory.

Allocating memory

Instead of `malloc` we have the keyword `new`, this returns a pointer to a memory address where the new block of memory resides.

De-allocating memory

Once you have finished with the memory you are using you need to tell the computer that you don't need it anymore. This is done in C with the function `free`. In C++ we gain the keyword `delete`. For instance to free the two blocks of memory allocated in the previous section we would use `delete` in this way.

WARNING: It is important not to mix C and C++ memory management functions. For example don't write.

```
MyClass * AClass = new MyClass();
Free(AClass);
```

If you use `new` then delete the memory with `delete`. If you use `malloc` or `alloc` then delete the memory with `free`.

Classes

Classes are an essential part of object oriented programming. Basically OOP is all about creating models of real life objects and using these within your program. For instance if you were creating a program that imitated a lighting system, you might want to create a light model, a switch model, maybe a timer model or light sensor model. These models are called objects.

So what does an object consist of? It consists of data, things the object 'knows' or attributes of the model, it also consists of methods, the way an object acts or what it can do. For example a child model may have the data `HairColour`, `EyeColour`, `Age`, `Height`

and Weight. In addition it can do things like walk, talk, eat and sleep. Obviously this is a very basic model of a child.

One advantage of using objects is a thing known as encapsulation. All the data and methods relating to an object is contained within it. It is possible to arrange things that only the object can access or alter its data. Looking again at the child example, you might arrange for a method called Birthday to increment the child's age by one. It would not make sense for the age to be incremented by any object other than the child.

There are other benefits such as inheritance which we will discuss later.

Basic Classes

So what does an object in C++ look like? For a start objects in C++ are called classes. We will start with a very simple example based on our child model.

```
class Child
{
    public:
        string HairColour;
        string EyeColour;
        int Age;
        int Height;
};
```

If you are thinking “Hey! that looks just like a struct”, then you would be correct. At this point there are two differences the first is the use of the keyword class instead of struct. The second is that keyword public:. There are three such keywords public:, protected: and private:. At this point we will consider the first and last of these, leaving protected: to our discussion of inheritance later on.

The keyword public: means that all data and methods that follow this can be accessed by other objects or from within your program. The keyword private: means that only the class can access this data or methods. Before continuing I had better start using some of the correct terminology. The data and methods contained in the class are called members, therefore the class has member data and member functions (or methods). To illustrate the difference between the public: and private: keywords let us consider how to access the member data in a class. First we need to create an instance of our class, then we can access its members. This is similar to using a struct.

```
Child MyChild;
MyChild.Age = 5;
```

This would work with the above class and a program using the above code would compile ok. However if we now change the class declaration to

```

class Child
{
    string HairColour;
    string EyeColour;
    int Age;
    int Height;
};

```

If you tried using this with the above code you would get a compile time error, why? By default all members are private so the above class is equivalent to

```

class Child
{
    private:
        string HairColour;
        string EyeColour;
        int Age;
        int Height;
};

```

So trying to use `MyChild.Age = 5;` is an error because only the class can access the member data `Age`, not external code which is what we are trying to use. It is good programming practice to declare member variables as private since it prevents external code from accidentally modifying these values. You may be asking at this point well how do I modify the values? The answer is by writing accessors (sometimes called setter and getter functions. If we do this then above class now looks like this.

```

class Child
{
    private:
        string HairColour;
        string EyeColour;
        int Age;
        int Height;
    public:
        void SetHairColour(string hairColour){HairColour = hairColour;};
        string GetHairColour(){return HairColour;};
        void SetEyeColour(string eyeColour){EyeColour = eyeColour;};
        string GetEyeColour(){return EyeColour;};
        void SetAge(int age){Age = age;};
        int GetAge(){return Age;};
        void SetHeight(int height){Height = height;};
        int GetHeight(){return Height;};
};

```

Now we can write `MyChild.SetAge(5);` and the code will work without errors.

Inheritance

Friends

Polymorphism

Typecasting

Type identification

Input/Output

In C we saw that everything is regarded as a file. C++ takes a different approach to input and output and regards everything as a stream. Using the power of classes an abstract stream class is defined and all input/output streams are derived from it. This means that they all share a common interface. The best news is that they are 100% easier to use than C's printf/scanf functions.

Input

Output

Strings

Namespaces

Exceptions

Templates

Chapter 5 – Finding your way around the IDE

To be written

Chapter 6 – Debugging with wxDev-C++

Some time ago I read an article that provided the mathematical proof that every computer program contains bugs. Back then I was still fairly idealistic about programming and I thought surely not. Now I agree, all but the most trivial programs contain bugs. Even the operating system you are using to run the program to read this contains bugs.

I'm not going to go into the history of why bugs are called bugs. I'll leave that as homework. Instead I will go into the genus of bugs. There are three main types. Some will cause your program to fail during the compile phase. Some will cause it to fail during the link phase and some will show up when the program runs.

These bugs come in roughly three categories critical, medium and low. Critical bugs are those that cause your program to wipe all the data from your user's hard drives. Medium bugs are those that crash the program occasionally. Low bugs are those that may irritate users but cause no real problems, like a text control that is one pixel lower than the others.

In any professional application far more time is spent fixing bugs than any other area of program development. One thing that all bugs have in common is that they can be hard to trace, especially for those that only show up at runtime. The purpose of this chapter is to show you some ways for tracking down and squashing those bugs. We will especially look at the tools provided by wxDev-C++.

Compile Time Bugs

Link Time Errors

Run Time Errors

DevC++ Related FAQs

Q. What is this I hear about an Easter egg in Dev-C++?

- A. One of the original developers created an Easter egg in version 4 of Dev-C++. This has survived in a slightly different form in the latest version and in wxDev-C++. To access the Easter egg in the latest version

Start Dev-C++ (or wxDev-C++).

Open the about box, accessed via Help|About Dev-C++...

Click on the picture at the top of the dialog and drag it down over the 'Authors' button.

A fish will appear moving across the dialog.

Depending on your speed and the speed of your computer you can try clicking on the fish. If you manage the fish will change direction.

Q. Why is my program not recompiled when one of the header files is altered?

- A. Normally if a source file is changed the compiler will pick this up and recompile the project. This is caused by setting the option in 'Use fast but imperfect dependency generation' in Tools|Compiler Options. This speeds up compilation of the project by missing out some steps, but causes changes in the header files to be missed.

The long term solution is to uncheck this option. If changes to header files are rare you can choose to recompile the whole project this will include the changed header files.

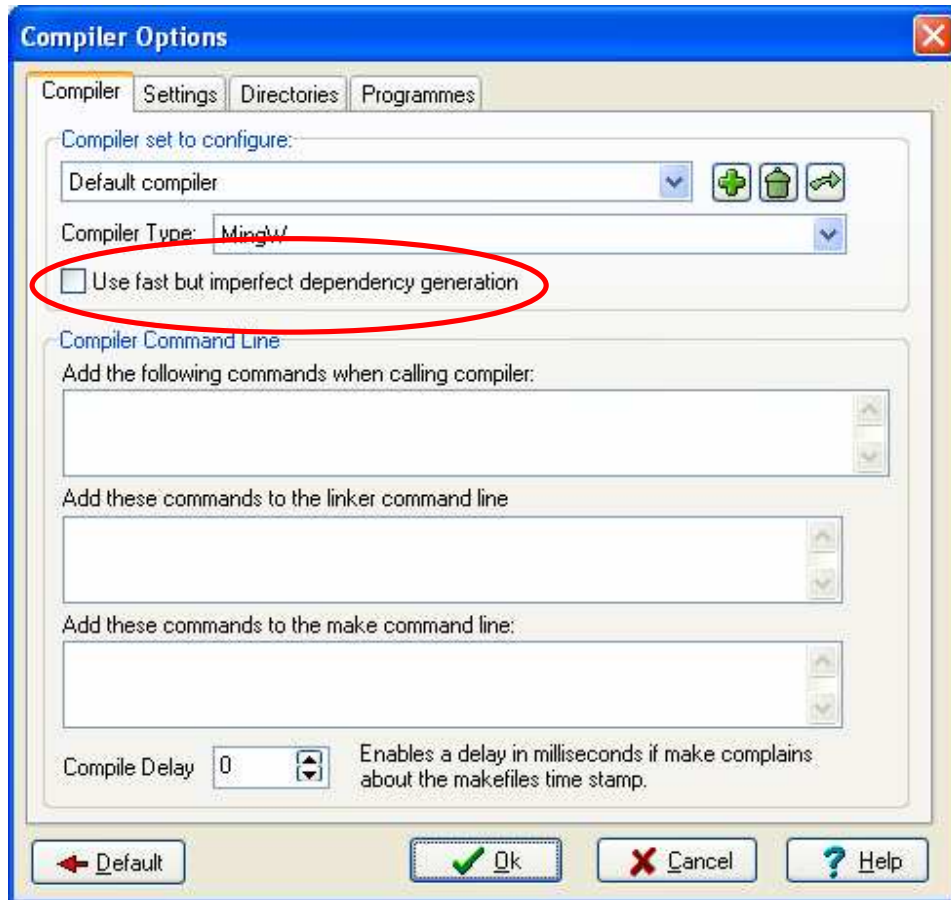


Figure x.x – Use fast but imperfect dependency generation

Part 2

Basic Development with wxWidgets and wxDev-C++

Chapter 7 – Creating a Basic HTML Editor

Introduction

So far we have covered C and C++ programming using just the DevC++ features of the IDE. But the features added by Guru Kathiresan and the other wxDev-C++ developers have created a RAD (Rapid Application Design) application of great power.

Part 2 takes you through using wxDev-C++ to develop programmes and introduces you to the various aspects of using it to design your own programs that look and act as you wish. In this chapter we will begin by designing a small HTML editor. This is nothing fancy, there is no syntax colouring and the like. What we will create is a programme with a basic menu, status bar, and the ability to save and load files. The special feature of this program is that it has two resizable windows. The bottom window is where you type html formatted text, the top window will update dynamically to show what this would look like in a web browser.

So let us start.

Starting a wxWidgets Project

We already know how to start a new project, so use your favourite method.

From the ‘New Project’ dialog, choose ‘wxWidgets Frame’.
Make sure that the ‘C++ Project’ radio button is selected
Enter ‘HTMLEditor’ as the project name.

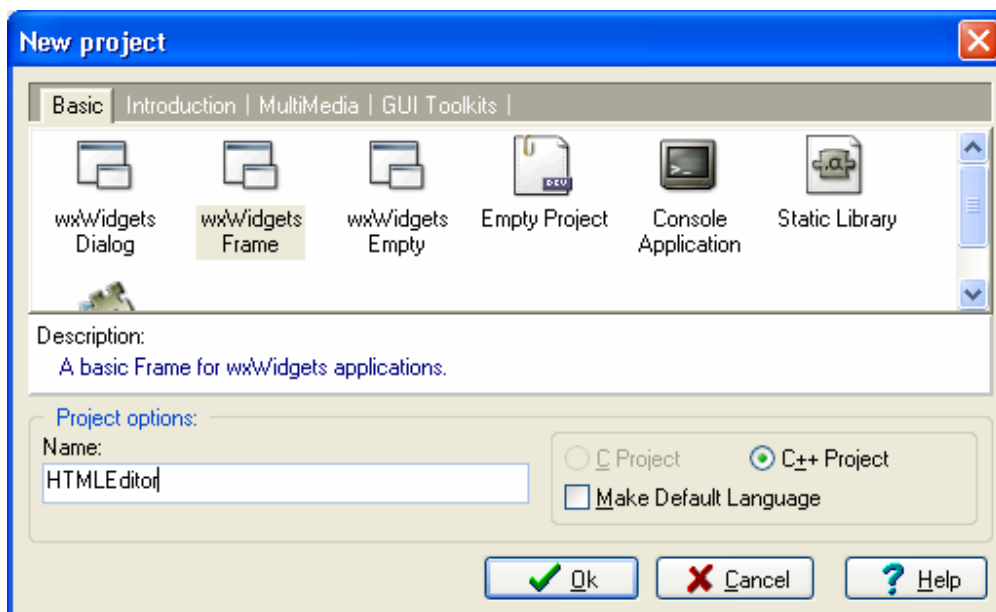


Figure 7.1 – Project settings for HTML editor

Create a new folder called 'HTML editor' within your 'Projects' folder.

You will be greeted with the unfamiliar 'Create New Project' dialog. For now alter the settings as follows, except you can put your own name where it says 'Sof.T'.

Then click on the [Create] button.

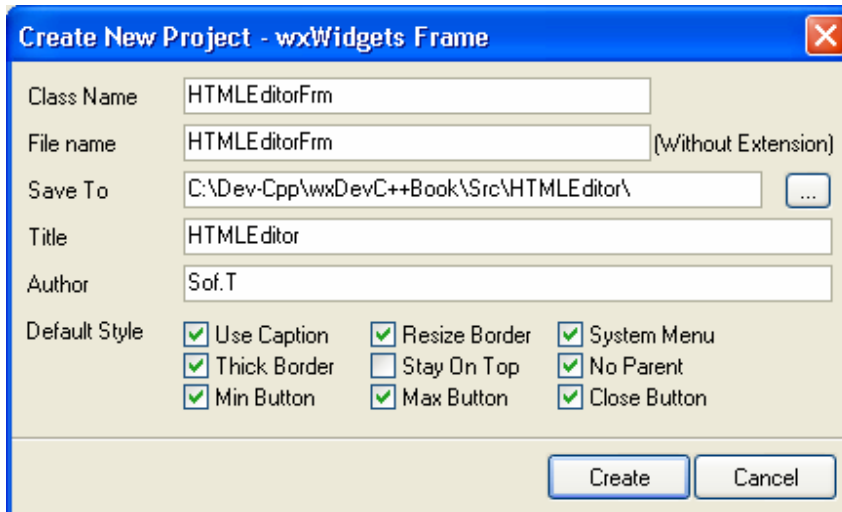


Figure 7.2 – Create New Project dialog

The IDE will flicker for a bit as various pages open, and finally the form designer will open. The IDE will look like this:

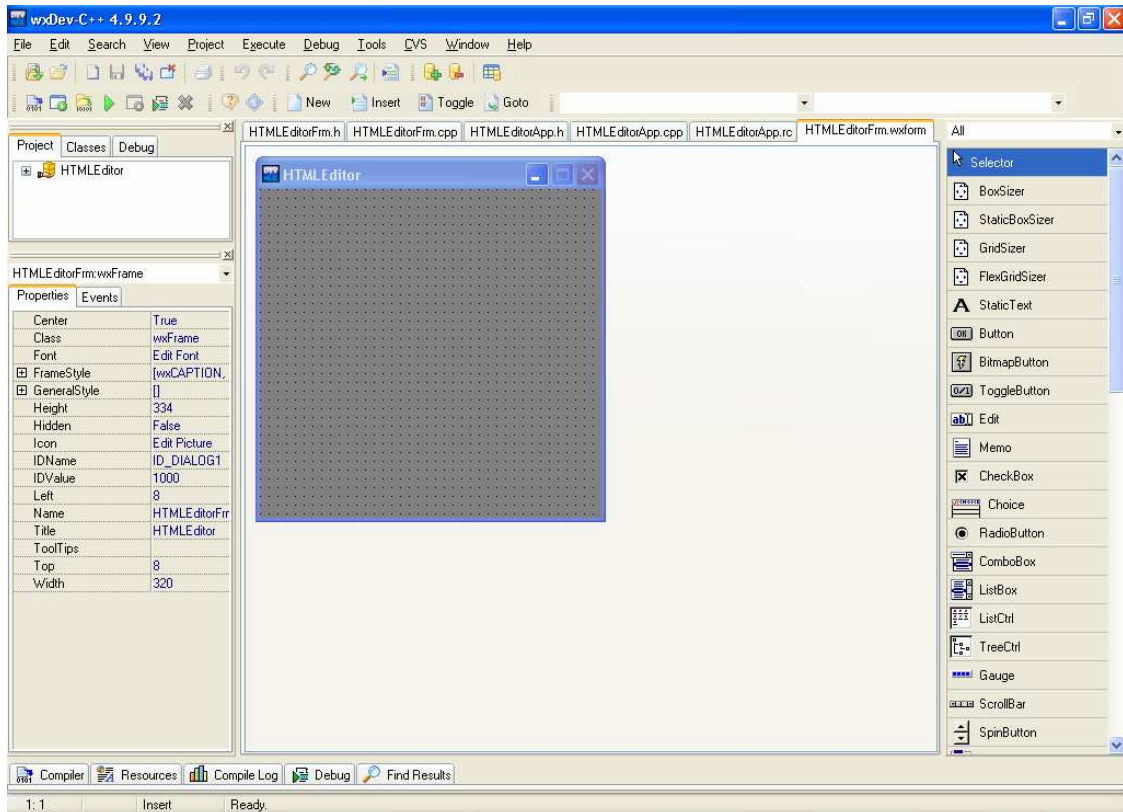


Figure 7.3 – The IDE showing the form designer

The following section will show you how to use this to create your killer application.

Using the Form Designer

In the centre is a dark grey window. This is the visual representation of your application. To the right of this is the palette showing the different components you can use. To the left is the property editor which you can use to change various aspects of your form. Just above this is a combo box which drops down to show all the components your application uses and allows you to quickly choose between them.

We will start by placing the components we need for our program. The total list will be a menu bar, status bar, splitter window, text control, and html control. So let us start by adding these. First will be the menu bar. What do you mean you can't see a menu bar?

If you look at the component palette on the right, you will see that it has a scrollbar on its right hand side. Using the scrollbar, many more controls become visible and you need to scroll down to locate them. Alternatively go to the combo box where it says 'All' and use this to reduce the number of components offered. Using either of these methods:

Find and select 'Menu'.

There, that is better. Now click on the menu bar component. This will now be highlighted in blue.

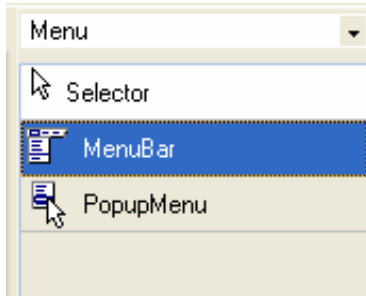


Figure 7.4 – Selecting a menu component

Click on the dark grey form in the centre of the designer.

You will notice two things: the first is that a small box with the picture of the menu bar has appeared on the form with 8 little squares around it. The other thing is that the 'Menubar' item on the component palette is no longer highlighted blue. The small box is a visual holder for the menu bar. Several components show up like this and it is nothing to worry about; it is quite normal and has nothing to do with the physical positioning of the component.

We will proceed by placing the other components we need onto the designer window. Next is the status bar. You will find this under 'Controls' and you may need to scroll down as it is near the bottom.

Click on 'status bar', and when it is highlighted, click on the form.

You will notice that this time it has automatically jumped to the right place on the bottom of the screen.

Now we want a splitter window. This can be found on the palette under 'Window'.

Find and select 'splitter window' and then click on the form.

The splitter window will automatically fill the form. Now we want a html window. This is on the same palette so:

Select 'HTML window' and drop it onto the splitter window.

You may find the form suddenly shrinks and scrollbars appear on the right and bottom sides, don't worry as this is a design feature. Finally we want a text control, which is found under controls. What we want is called 'Memo'. This is actually just a text control with multiple lines, the name is a part of wxDev-C++'s Delphi heritage.

Select the 'Memo' component and drop it onto the form.

You will find that the Memo control jumps to the left of the HTML control. This is not what we want. To sort this out click on the Splitter Window so the form looks like this.

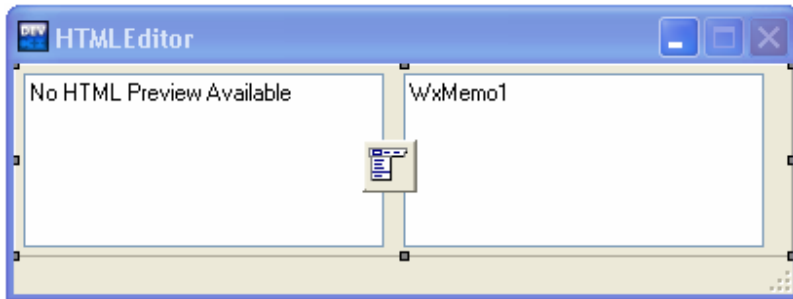


Figure 7.5 – The Design Form with the Splitter Window selected

We now need to find the Property Inspector on the left of the IDE

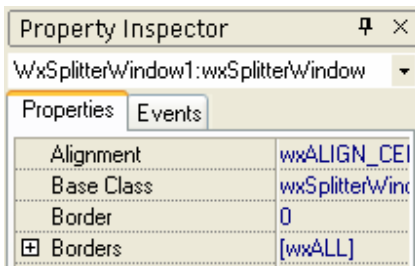


Figure 7.6 – The Property Inspector

Find the property Orientation



Figure 7.7 – The Orientation property

Change this to wxVertical.
Click on the Design Form to see the change.

We will learn more about using the Property Inspector in the next section. For now we have a form like the following.



Figure 7.8 – The completed form in the design window

Now for the moment of truth.

Press <F9> to compile and run the application.

You will notice that the compiled application looks somewhat like the design form. The text has disappeared from the html window but not from the text control. Also notice that although we dropped a menu bar on the form nothing shows up on the compiled application.

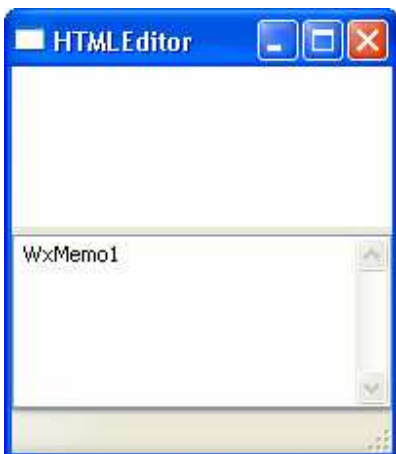


Figure 7.9 – The compiled application.

In the next section we will look at configuring the form in the designer to make it more like a real working application.

Altering Properties

If the application is still running, close it, just like any other windows application, by clicking on the red close button on the caption bar. Now we can start to configure our components in the form designer.

First we will correct that text control. In the form designer:

Select the text control by clicking on it.

You can tell if it is selected because the eight little boxes will appear on the sides and corners (the bottom ones may be hidden by the status bar). These boxes are called handles. You will note that when you select a component the text changes in the combo box above the Inspector. This text gives you the component name followed by the component type. E.g. when the text control is selected it changes to WxMemo1:wxTextCtrl. WxMemo1 is the controls name and wxTextCtrl is its type. The contents of the property editor also change as different components are selected, to reflect the properties each component contains.

So having selected the text control the Property Inspector should look like this.

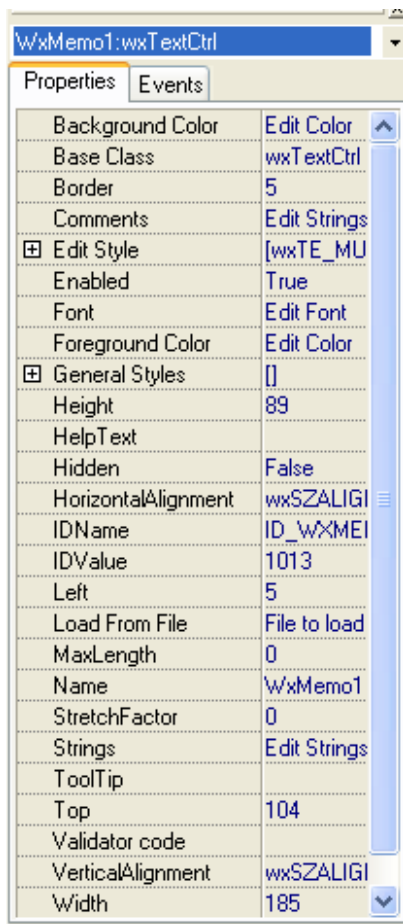


Figure 7.10 – The Property Inspector for the text control

First let us get rid of that text that shows up in the control. If we look at the property editor the first column contains the property names; these are in black and can't be altered. The second column with the blue writing contains the values for these properties and can be adjusted. At first it isn't easy to see what alters this. The property we need is under 'Strings'. If you

Click in the second column, next to 'Strings' where it says 'Edit Strings'

a small button with three periods in it will appear.

Click on this [...] button to activate the text editing dialog.

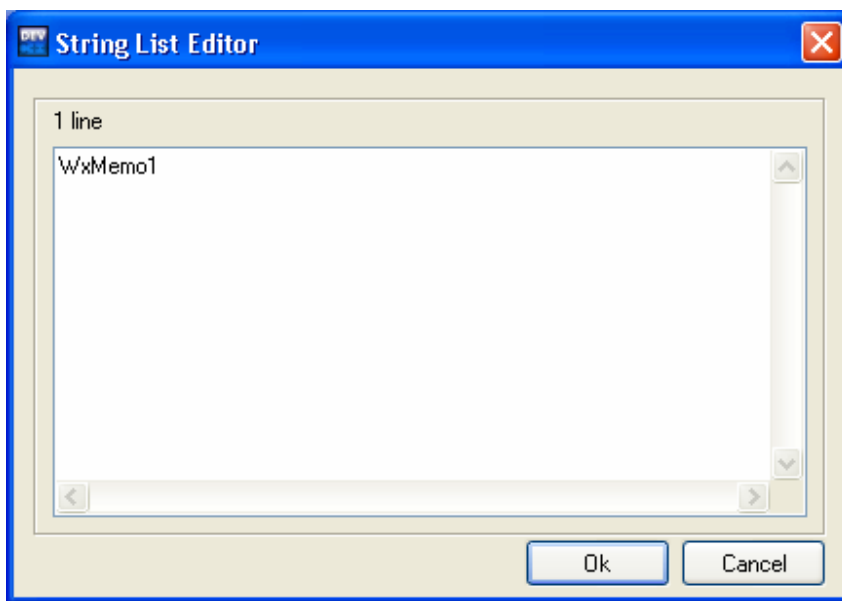


Figure 7.11 – The string editor

Any text shown in this editor will appear in the text control.

Delete all of the text control contents.
Press the [Ok] button.

You will notice that the text has disappeared from the text control in the form designer. If you want to check that it has also disappeared from the compiled application, you can compile and run it again.

Now let us sort that menu bar out. Why didn't it show up? The reason for this is that it doesn't contain anything yet. To correct this:

Select the menu bar by clicking on the small square that represents it.

This time the property editor only shows five items. The one we want is called 'Menu Items', just like the 'Strings' property in the text control clicking next to this produces a button with three periods in it.

Click on the [...] button to activate the following dialog.

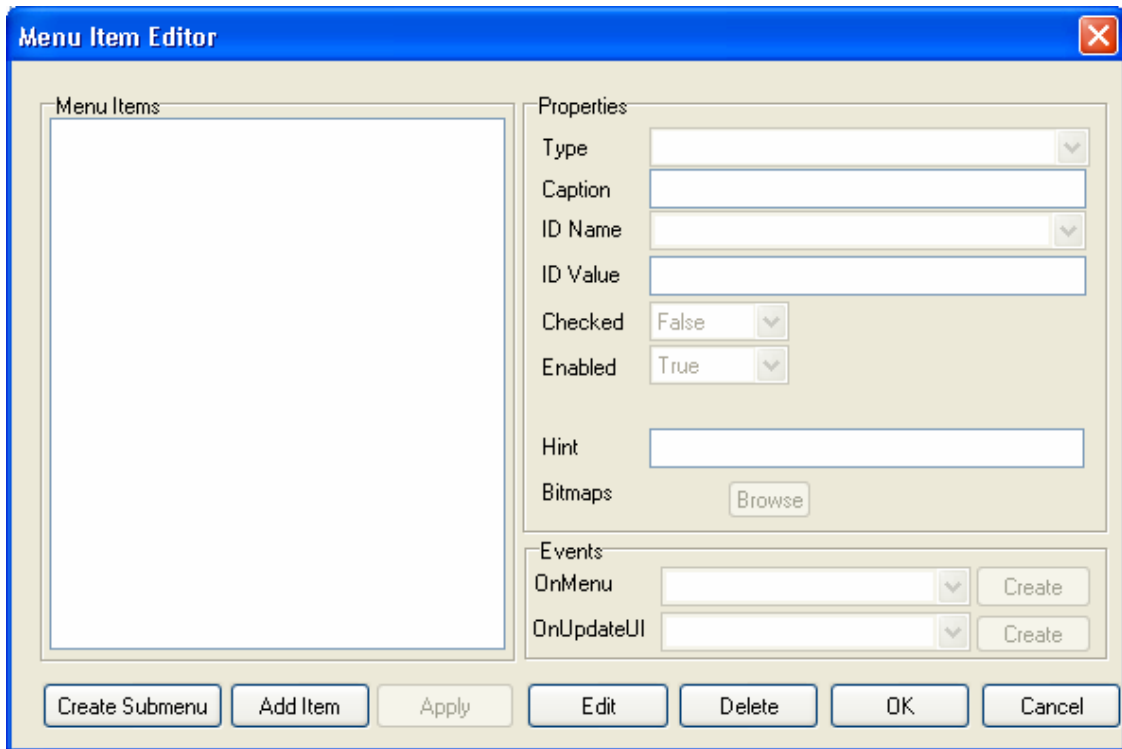


Figure 7.12 – The menu bar editor

Click the [Add Item] button.

Suddenly the dialog fills with all sorts of information that is generated for you. The box labelled 'Caption' now contains the value 'MenuItem1';

replace 'MenuItem1' with '&File' and press <Enter>.

The text now changes on the Menu Items box and the ID Name also changes.

Click on the [Apply] button.

(The reason for using '&' in the menu item names is to do with mnemonics and will be explained later in Chapter 12 under 'Mnemonics and Keyboard Accelerators').

Now we have created our File menu we want to add some items to it. To do this:

Click on the [Create Submenu] button.

This produces a menu item under the File menu and inset a little to show it is a child of the File menu.

Change the caption to '&Open\tCtrl+O' and press <Enter>.
Click on the [Apply] button.

Repeat this procedure this time changing the caption to '&Save\tCtrl+S'.

(The reason for using '\tCtrl+O' in the menu names is to do with keyboard accelerators and will also be explained in Chapter 11 under 'Mnemonics and Keyboard Accelerators').

Now we want to add a separator to the menu.

Click on the [Create Submenu] button.

This time in the top combo box labeled 'Type':

Select 'Separator'.
Click on the [Apply] button.

Finally we want to add an exit entry to the menu. For the last time:

Click on the [Create Submenu] button.
Alter the caption to read 'E&xit'.
Finally click on the [OK] button.

<p>WARNING: If at anytime you are creating a menu bar you click the [Cancel] button, all the work you have done up to this point will be lost. This can be extremely irritating when you have created a large complex menu system. If you have made a mistake it is better to continue. Click on the [OK] to save the work you have done so far, then reopen the menu bar dialog to make changes.</p>
--

To see the result of all our altering of properties:

Press <F9> to compile and run.

You will notice that we now have a menu bar. Click on File and you will see the newly created menu drop down. It doesn't do anything yet, but it will by the end of the next section.

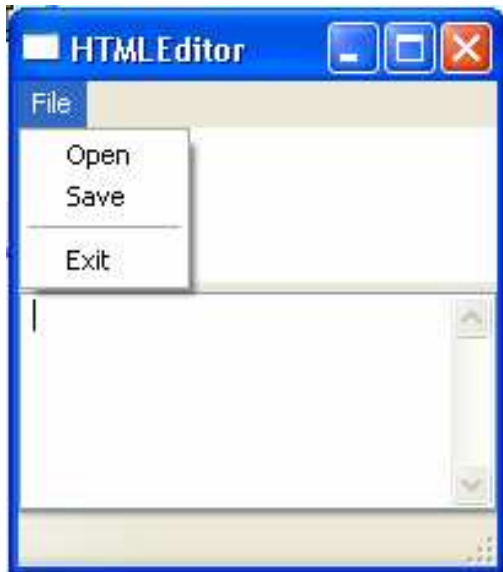


Figure 7.13 – Our new menu bar

Before we start coding we need to add two more components.

Change the component palette to 'Dialogs'.

Select and drop an OpenFileDialog and a SaveFileDialog on to the form.

After adding them to the form:

Select each in turn and alter the Property 'Extensions' from '*.*' to '*.htm'.

Now we can move on to the next part 'Adding Code' and actually get this to do something.

Adding Code

GUI programming relies on what is called 'Event based' programming. Basically once the program starts it just sits there in a loop, waiting for an event to take place. When an event does take place it looks to see what, if any, code it should execute in response. Events can be anything from starting a programme, to clicking within the programme window with a mouse, resizing the window or leaving it to interact with a different programme.

So to get our programme to do anything, we have to decide what events to respond to, and what to do in response. We will start with an easy one. At present our menu does nothing. Lets change that and make our programme close when we click 'Exit' in the file menu.

To do that:

Select the menu bar component.
Click on the [Edit MenuItems] button.

If the menu tree on the left hand side looks like this. Then click on the '+' sign next to it to show all the menu entries.

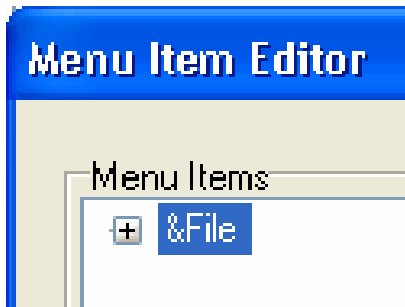


Figure 7.14 – Click the '+' sign to expand the menu tree

Select the 'E&xit' entry.
Click on the [Edit] button on the bottom of the dialog.

Now on the right of the dialog near the bottom look for the combo box labeled 'OnMenu'. This control holds the menu event that occurs when we click 'Exit' on the menu.

Click on the [Create] button (next to the combo box) .

The following dialog will pop up.

Click on the [OK] button.

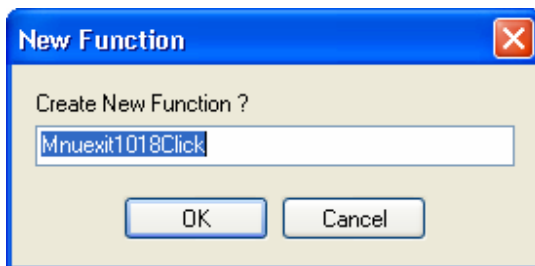


Figure 7.15 – The create new function dialog

Click the [Apply] button
Click the [OK] button.

If the designer does not take you directly to the new function then click on the tab labeled '(*)HTMLEditorFrm.cpp' (in the editor window). The page should open with the cursor in the body of a function as illustrated below.

```
95     * Mnuexit1018Click
96     */
97     void HTMLEditorFrm::Mnuexit1018Click(wxCommandEvent& event)
98     {
99         // insert your code here
100    }
```

Under the line

```
// insert your code here
```

add the line

```
Destroy();
```

The Destroy function will cause the application to close.

Press <F9> to compile and run.

Go to File|Exit to see that the code we added works.

Ensure the form designer has focus by clicking on the tab labelled 'HTMLEditorFrm.wxform' in the editor window.

We now want to add the functionality that creates the text editor part. Although not the best way to do this, we will update the HTML window every time the user alters or updates the contents of the text control. So it is an event within the text control we want to act upon.

Select the text control.

If you look at the property editor you will see another tab called 'Events' nestling behind it.

Click on the 'Events' tab to discover what events the text control can respond to.

You should see something like the following. Remember we want to act when the user updates the text control. So I guess the event 'OnUpdated' is the one for us.

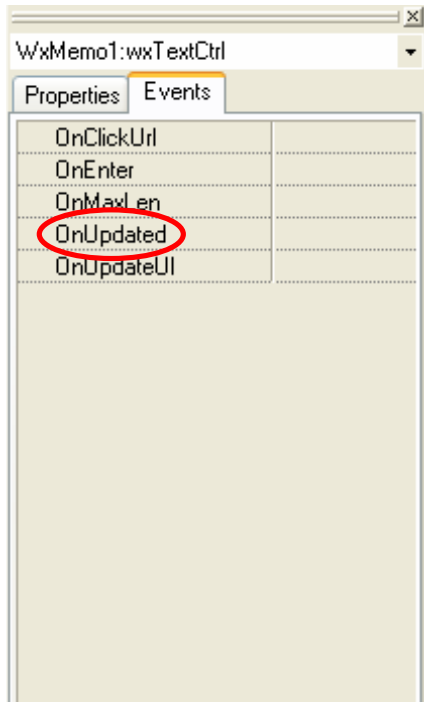


Figure 7.16 – The event editor

To create a new function to respond to this event,

Click in the empty cell in the right hand column.

A drop down list box should appear listing all the functions wxDev-C++ has created for us already. We don't want to use any of these so:

Select the top option <Create New Function>.

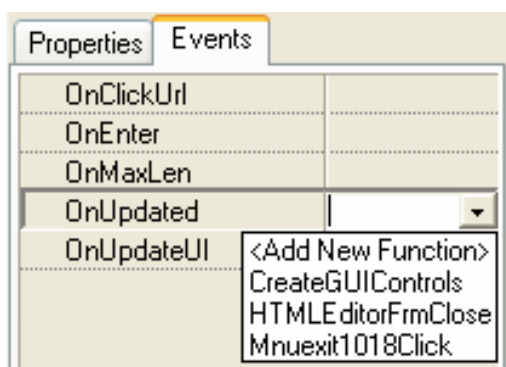


Figure 7.17 – List of available functions in the event editor

You should be taken automatically to the new function, but if not, click on the tab labeled '(*)HTMLEditorFrm.cpp'. There you will see a new function similar to the following.

```

106     * WxMemolUpdated
107     */
108     void HTMLEditorFrm::WxMemolUpdated(wxCommandEvent& event)
109     {
110         // insert your code here
111     }

```

wxDev-C++ has kindly created the function skeleton for us, but what are we going to add inside it. This time I am not going to tell you straight out. I want you to learn to make use of one of wxDev-C++'s greatest provisions, the wxWidgets help file. So in the IDE go to:

Help|wxWidgets
Or just press F1.

The help file should appear:

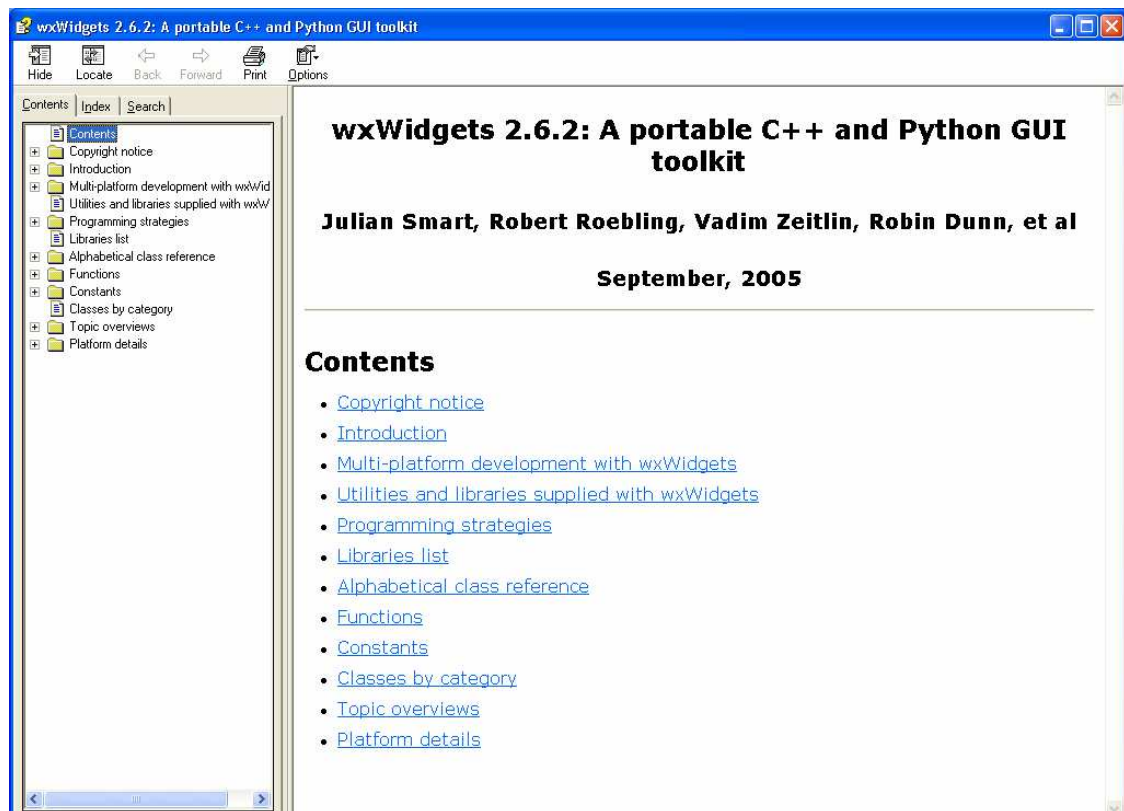
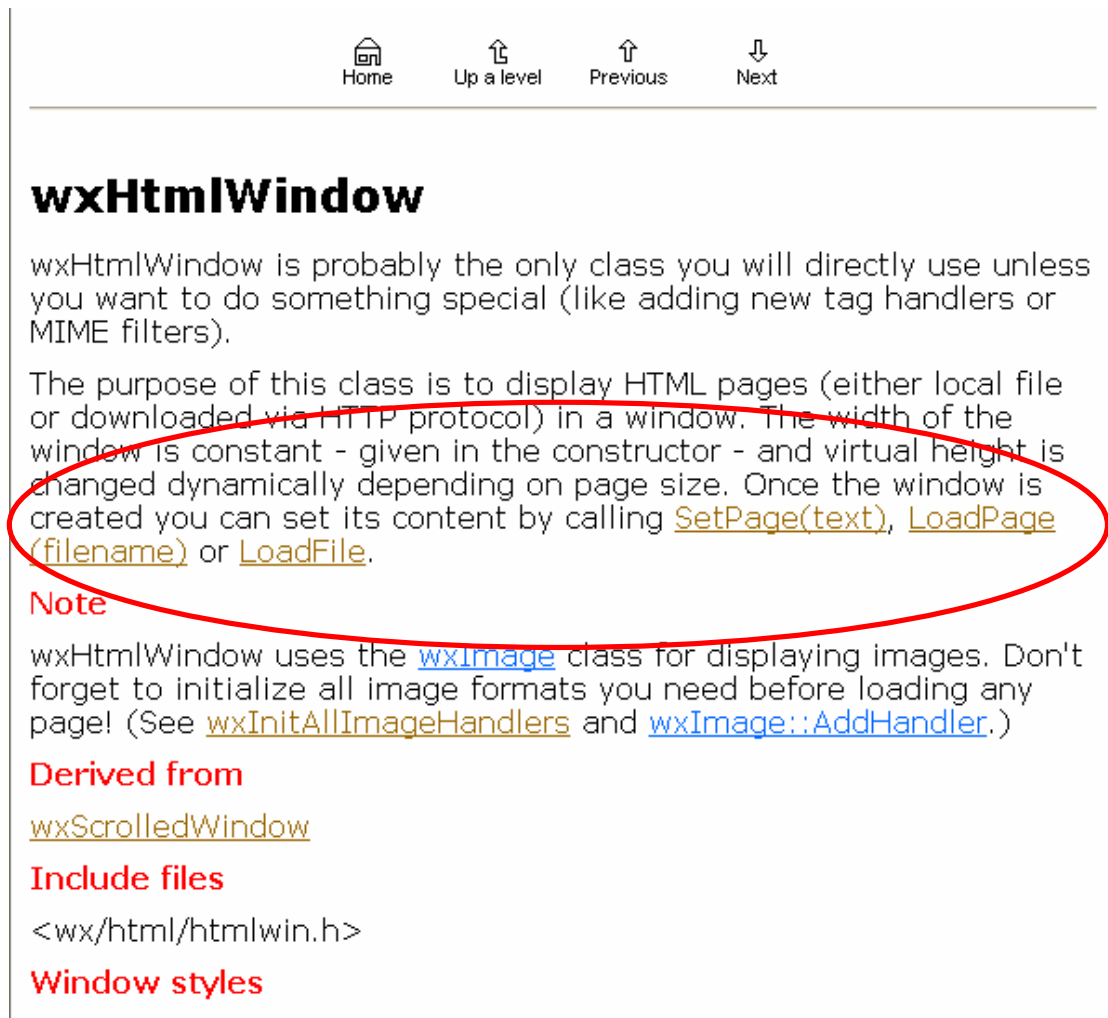


Figure 7.18 – The wxWidgets file help, this should be your greatest friend

Our objective is to alter the contents of the HTML window based on the contents of the text control. So let us first find out how to alter the HTML window. Under the 'Contents' tab on the left hand panel of the help screen, expand the option 'Alphabetical class reference' by clicking on the '+' next to it. A long list of components all beginning with 'wx' will be displayed. How do we know what one to look for? Well at a rough guess let us scroll down to wxHtml. There are several entries beginning with this, but look closely

and there, near the end, is our one, 'wxHtmlWindow'. Click on the entry 'wxHtmlWindow' and read the text that fills the right hand pane.



The screenshot shows a documentation page for wxHtmlWindow. At the top, there are navigation links: Home, Up a level, Previous, and Next. The main heading is 'wxHtmlWindow'. The text describes the class and its methods. A red circle highlights the sentence: 'Once the window is created you can set its content by calling [SetPage\(text\)](#), [LoadPage\(filename\)](#) or [LoadFile](#).' Below this, there are sections for 'Note', 'Derived from', 'Include files', and 'Window styles'.

Home Up a level Previous Next

wxHtmlWindow

wxHtmlWindow is probably the only class you will directly use unless you want to do something special (like adding new tag handlers or MIME filters).

The purpose of this class is to display HTML pages (either local file or downloaded via HTTP protocol) in a window. The width of the window is constant - given in the constructor - and virtual height is changed dynamically depending on page size. Once the window is created you can set its content by calling [SetPage\(text\)](#), [LoadPage\(filename\)](#) or [LoadFile](#).

Note

wxHtmlWindow uses the [wxImage](#) class for displaying images. Don't forget to initialize all image formats you need before loading any page! (See [wxInitAllImageHandlers](#) and [wxImage::AddHandler](#).)

Derived from

[wxScrolledWindow](#)

Include files

<wx/html/htmlwin.h>

Window styles

Figure 7.19 – Looking through the description of the wxHtmlWindow

As we read through the description of this control we reach the sentence which says, "Once the window is created you can set its content by calling SetPage(text), LoadPage(filename) or LoadFile.". Now to me the last two options tell me that they open files, whereas the first one 'SetPage(text)', says that we can set the contents of the HTML window by calling this function with some text. To confirm this click on the link SetPage(text) and read more about it.

So now we can add SetPage(text) into our new function. First we need the name of the control we are working with. To get this go to the form designer and hover over the HTML control. A pop up should appear saying 'WxHtmlWindow1:wxHtmlWindow'. Remember the bit before the colon is the component name. So go back to our function and under the line:


```
// insert your code here
```

add the code

```
WxHtmlWindow1
```

WxHtmlWindow1 is a pointer to an instance of HtmlWindow, so we need to add -> finally add the new function we found in the help file

```
SetPage().
```

The completed command should look like:

```
WxHtmlWindow1->SetPage()
```

So far so good, but we now need some text as an argument to the function.

We want to get the text from the text control to pass to the HtmlWindow. Try scrolling down the list and look for 'wxMemo'. Having trouble? Remember what I said earlier about 'Memo' being a hangover from Delphi? To find what to look for, try the trick of hovering over the component on the editor.



Figure 7.20 – Discovering the name and type of a control

The format is <Control Name>:<Control Type>, so the section after the colon is the information we want. Look in the help file for 'wxTextCtrl'. When you click on wxTextCtrl there is very little information given to you. Expand the left hand tree list of contents, by clicking on the '+' sign next to wxTextCtrl. Now you can see all the functions contained in the component. Look down the list for suitable candidates. We are looking for something most likely starting with get. Since there is no 'GetText' or anything like it, the nearest appears to be 'GetValue'. Click on this and read about it to make sure. The first line should convince you. 'Gets the contents of the control.'

To add this to our function, inside the braces ‘ () ’ of ‘ SetPage ’, add the name of the text control, and remembering it is a pointer, add the new function we found. You should end up with this code.

```
108  /*
109  * WxMemo1Updated
110  */
111  void HTMLEditorFrm::WxMemo1Updated(wxCommandEvent& event)
112  {
113      // insert your code here
114      WxHtmlWindow1->SetPage(WxMemo1->GetValue());
115  }
```

So let us carry on coding. What do you mean you want to see what this has done. Very well, let us compile and run it first.

Press <F9>.

If you don't know about HTML tags then enter the following: ‘<h1>wxWidgets are great</h1>the designers of wxDev-C++ are <u>GREAT</u> too!’ . You may want to expand the window to see the full effect. Or if you have downloaded the sample code to accompany the book there is a file called ‘sample.htm’ included for you to open. You will need to maximize the window to see the result.

Now can we carry on? We have two more functions to create. One to load HTML files and one to save HTML files. In preparation for this we created two menu entries called “Load” and “Save”. The actual work will be carried out by the Open/Save File Dialogs we dropped on the form earlier. We will create the functions we need as we did for the “E&xit” menu entry.

Open the menu dialog editor by selecting the component that represents the menu bar.

Click on the ‘Edit MenuItems’ cell.

Expand the tree list of menu entries.

Select the entry “&Open”.

Click on the [Edit] button.

Next to the ‘OnMenu’ combo box, click on the [Create] button.

On the dialog that pops up

Choose the [OK] button,

Click on the [Apply] button.

Repeat the same routine for the “&Save menu” entry.

When you have completed this

Click the [OK] button.

The following code should have been generated for you.

```
116  /*
117  * Mnuopen1015Click
118  */
119  void HTMLEditorFrm::Mnuopen1015Click(wxCommandEvent& event)
120  {
121      // insert your code here
122  }
123
124  /*
125  * Mnusave1016Click
126  */
127  void HTMLEditorFrm::Mnusave1016Click(wxCommandEvent& event)
128  {
129      // insert your code here
130  }
```

Alter these functions to look like the following. I will explain more about this code in Chapter 9 in the section titled “Dialogs”. But I would recommend opening the wxWidgets help and looking at the components and functions used in the following code to understand how they work.

```
116  /*
117  * Mnuopen1015Click
118  */
119  void HTMLEditorFrm::Mnuopen1015Click(wxCommandEvent& event)
120  {
121      // insert your code here
122      if(WxOpenFileDialog1->ShowModal())
123      {
124          WxMem01->LoadFile(WxOpenFileDialog1->GetPath());
125      }
126  }
127
128  /*
129  * Mnusave1016Click
130  */
131  void HTMLEditorFrm::Mnusave1016Click(wxCommandEvent& event)
132  {
133      // insert your code here
134      if(WxSaveFileDialog1->ShowModal())
135      {
136          WxMem01->SaveFile(WxSaveFileDialog1->GetPath());
137      }
138  }
```

That is it for this programme, so press <F9> to compile and run. Try loading and saving ‘*.htm’ files. Alter the text in the lower text control and see how this is reflected in the upper HTML control.

By now I hope you are impressed by what wxDev-C++ and wxWidgets can do for you. The amount of code you have written is comparable to the C and C++ samples and they did very little. This program achieves so much more, with no more effort.

You may wish to extend this programme to have a toolbar, or buttons to insert various HTML tags, or other amazing features. If you do want to, then go ahead. If you need to know more about how to achieve this, then read on.

Chapter 8 – Working With Frames and Dialogs

Introduction

The basis of any GUI program no matter how simple is the window or windows that you will use. wxWidgets makes a distinction between Frames and Dialogs, although they can for the most be used interchangeably. Dialogs generally are used for allowing users to make choices or answer questions. It is usual for dialog to be non resizable and to be displayed Modally. A modal dialog means that the user cannot interact with any other frames or dialogs until this one is closed. Frames are generally used for the main application windows containing menus, toolbars, status bars and the like.

In this chapter we will look at creating new projects and the difference between Frame and Dialog projects. We will look at the different property options for frames and dialogs. We will also look at how to add more frames and dialogs to an existing project. Finally we will start to build a sample application.

Before we continue it is necessary to cover a point which causes confusion for some new users. wxDev-C++ has been designed so that it can be used as a designer of forms and dialogs for use outside of a project. As a result users are presented with the following dialog when they start a new project.

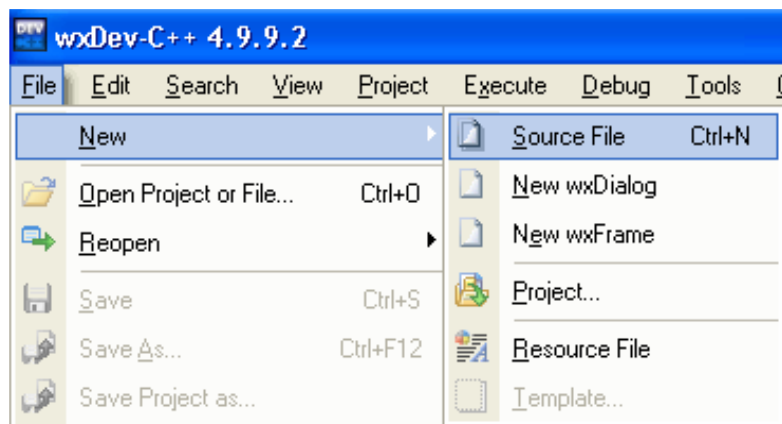


Figure 8.1 - The New menu for creating new projects, source files and forms.

Many users have clicked on the option “New wxFrame” and designed a nice GUI. However they are surprised when they try to compile since they get a load of linker errors. The reason for this is that there are no project settings so the linker doesn’t know where to look for the wx libraries it needs. The menu options New wxDialog and New wxFrame are there to either add new frames and dialogs into an existing project or to design frames and dialogs for use elsewhere. To create a new project containing a frame or dialog you need the “Project...” option.

Creating a New Project

This is where it all begins. So far we have created a number of new projects and should be getting familiar with some of the options available. Mostly these projects have been command line programs with the exception of our HTML editor example. So let us get started with how to create new frame or dialog projects and the difference between them.

Frame Project

In this example we will create a basic 'Hello World' program. To start

Select the menu File|Project

The following dialog will be displayed

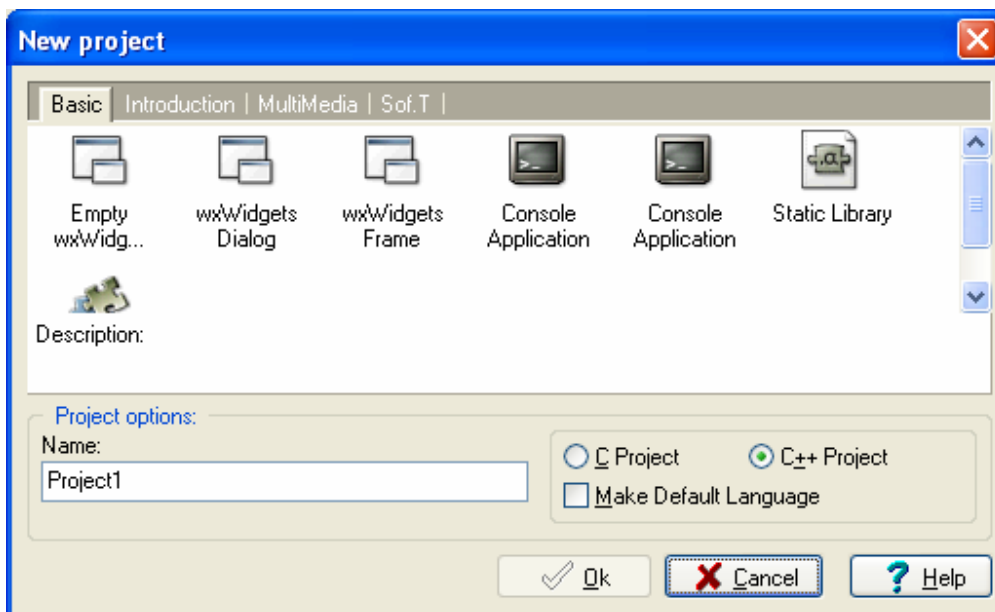


Figure 8.2 – The new project dialog

Clicking on each of the icons will give you a brief description of what the project type includes. Since we want to create a frame project we need to carry out the following steps.

- Click on wxWidgetsFrame.
- Change the project name to FrameProject.
- Make sure that the radio button next to C++ Project is selected.
- Click on the [Ok] button.

This will result in the dialog closing and a new dialog being opened up which looks like the following.

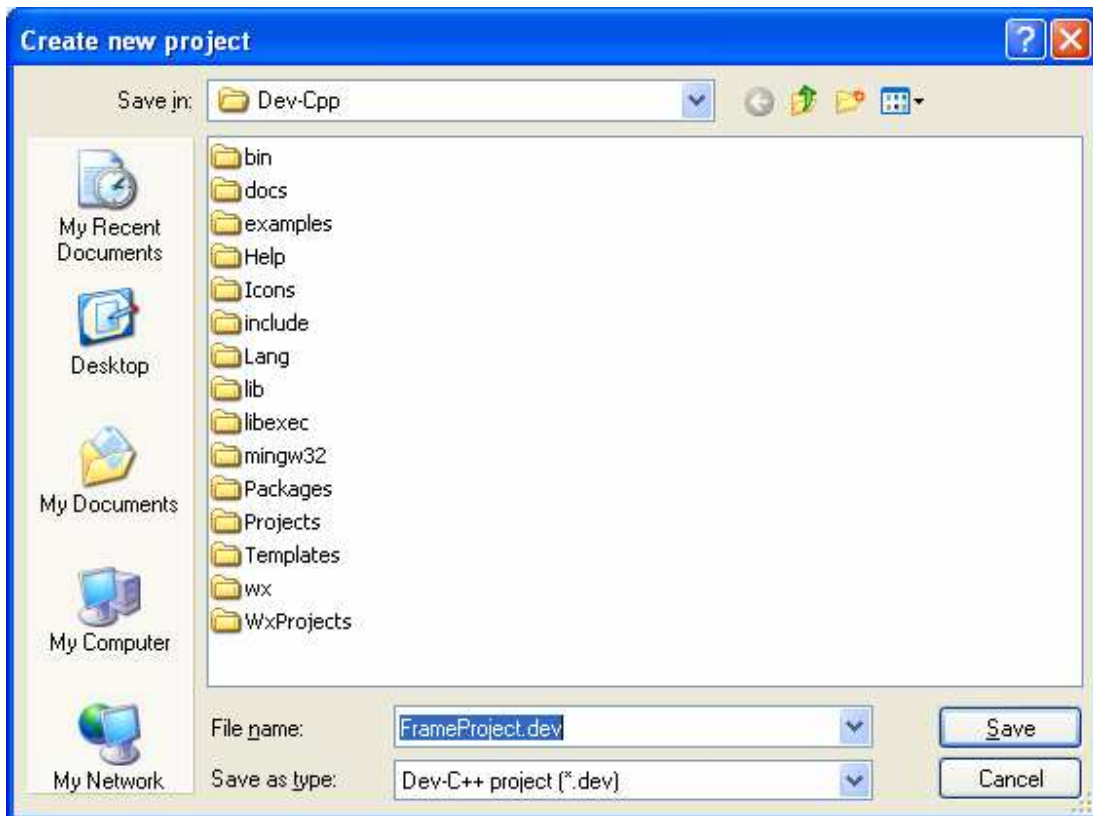


Figure 8.3 – The save project dialog

Just as we have seen before this dialog gives us the option to save our project to a specific locate. If you have been following all the tutorials in this book

- Go to your project folder
- Create a new folder called FrameProject
- Enter this folder
- Click on the [Save] button

After the save dialog closes a new dialog will open.

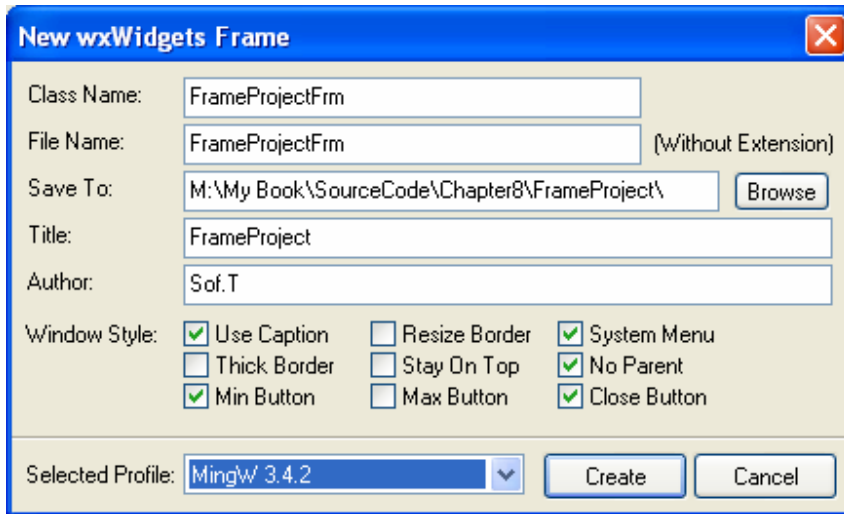


Figure 8.4 – The new frame dialog

There are a lot of options here so let us look at them one by one.

Class Name

Since every new frame you create is derived from the wxFrame class it has to be given its own class name. This is automatically generated for you, but you have the option to change it. If you change it there are certain rules to remember. A Class name can only contain alpha-numeric characters and underscores. In addition the class name must not be a keyword, must be unique and cannot begin with a number.

Let us try to alter the class name to an invalid name.

Enter “2 Frame ProjectFrm” in the class name field
Click the [Create] button

A warning dialog is displayed like the following

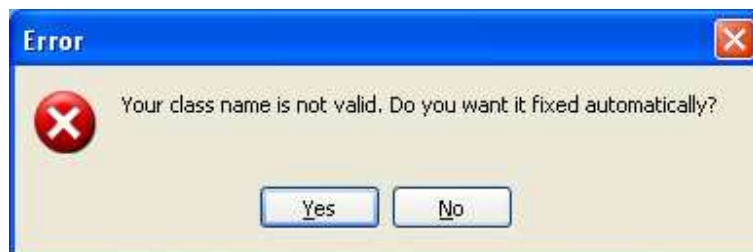


Figure 8.5 – Invalid class name warning dialog

If you click [No] a remainder pops up to alert you to the problem.

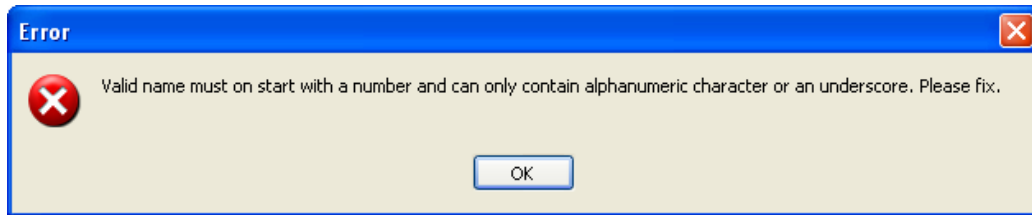


Figure 8.6 – Dialog to remind you what a valid class name can contain.

If you choose yes wxDev-C++ replaces all invalid characters with underscores, this may make you class name look strange, but at least the compiler will not complain.

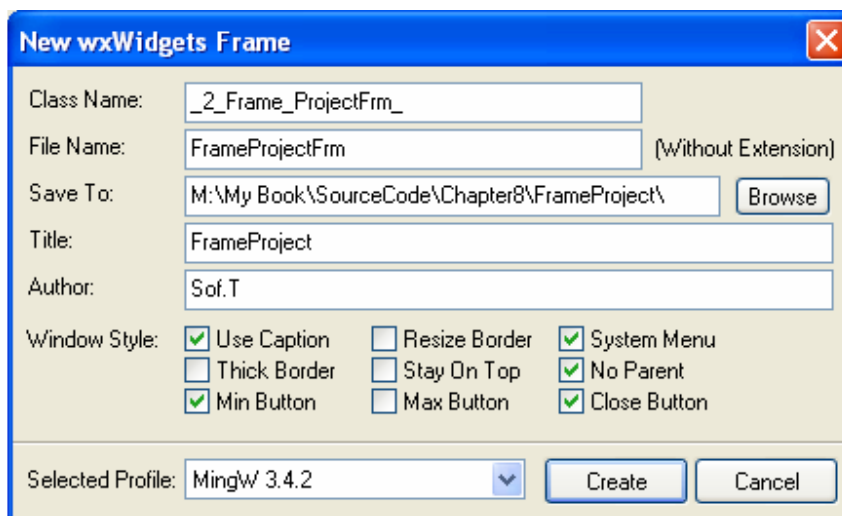


Figure 8.7 – An automatically corrected class name.

For this example we want the default value for the class name so

Fill the Class Name field with FrameProjectFrm

File Name

This field holds the name of the file that the information about your frame class will be saved in. Every frame or dialog created by wxDev-C++ is saved in separate files. In fact there are three files for each frame or dialog. They all have the same name only the extension is different. One ends in .wxform this is the information needed by wxDev-C++ to store what your file looks like in the designer. Then there are two source code files the header file ends in .h and contains the declaration of your frame class and the other ends in .cpp and contains the definition of your class.

Once again you can alter this to suit your own purposes. Should you choose an invalid file name wxDev-C++ will warn you and offer to automatically correct this. For this exercise we will accept the default value.

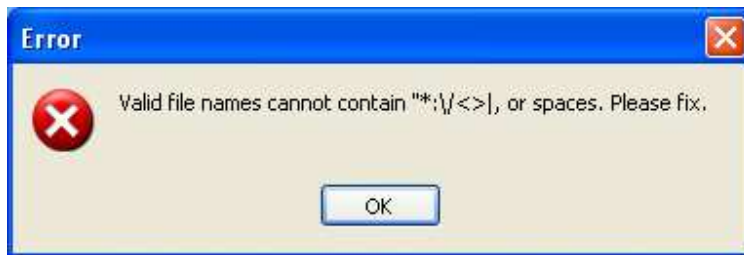


Figure 8.8 – Dialog to remind you what constitutes a valid file name

Save To

This field contains the location you elected to save the project to in the previous dialog. It is generally best to leave this field alone unless you have a definite reason to save the code for your frame in a different location.

Title

The contents of this field are used as the caption for this frame. The caption is the written part on the top of the border.



Figure 8.9 – The caption part of a frame.

For this example we will leave the caption alone.

Author

This field contains your name. wxDev-C++ will try to determine this name automatically. The contents of this field are used for adding copyrighting information to the source code and project files. If this field is incorrect or you want to use different copyright information change this to suit.

Window Style

This section contains a variety of check box which alter the look and behaviour of your frame. Some of these are filled by default. We will look at each of the options in turn and see what they mean.

Use Caption

This option corresponds to the flag `wxCAPTION`. The help documentation says that setting this option puts a caption on the frame. In fact under windows if you don't have this set your frame will display without a border.

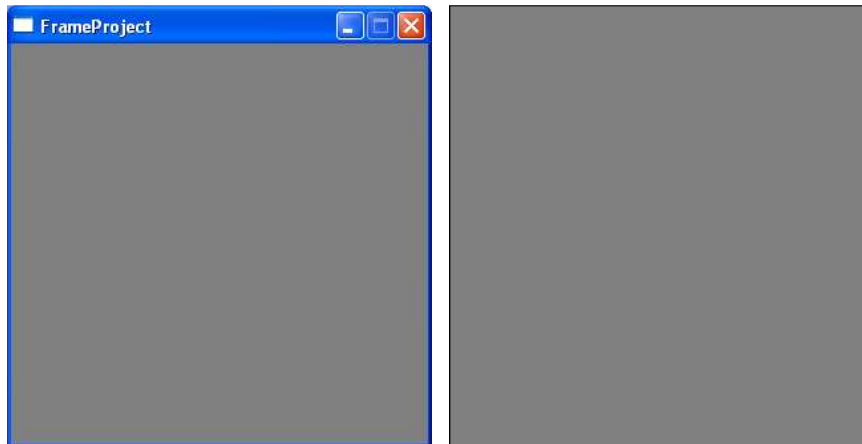


Figure 8.10 – Difference between a frame with caption set or not.

We will leave the caption option checked.

Resize Border

This option corresponds to the flag `wxRESIZE_BORDER`. This option means the user can drag the edges of the frame to make it larger or smaller.

We will select this option by ticking it.

System Menu

This option corresponds to the flag `wxSYSTEM_MENU`. This determines if the icon is shown in the top left of the windows. If it is shown then a menu is available by clicking on it. This option also affects whether the minimize, maximize and close buttons are shown or not.

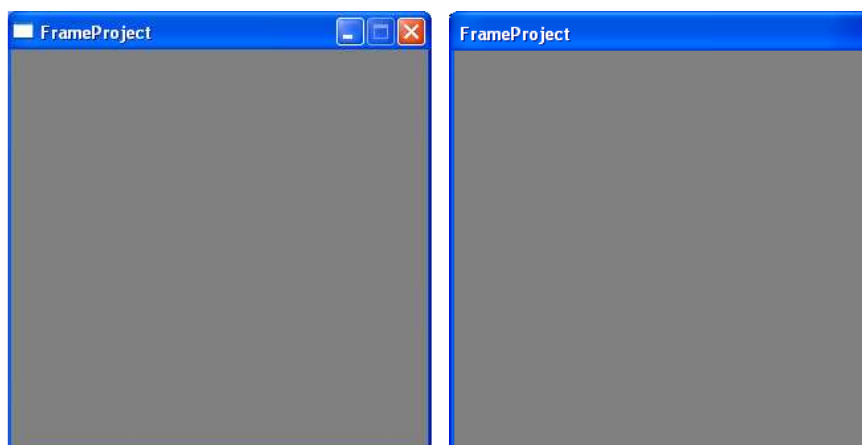


Figure 8.11 – Difference between a frame with system menu set or not.

We will leave this option selected.

Thick Border

This corresponds to the flag `wxTHICK_FRAME`. Which the documentation tells us display a thick frame around the window on Windows and Motif applications. Personally I have never seen this make a difference on Windows applications. It appears this flag relates to the Windows flag `WS_THICKFRAME` which has had little meaning since the very early days of Windows.

We will leave this setting as it is.

Stay On Top

This option corresponds to the flag `wxSTAY_ON_TOP`. This only works on Windows at the present and means that this window will float above all other windows. This is useful if you are designing a window that will be used as a toolbox, or a dialog that the user must not ignore.

We will leave this setting as its default value.

No Parent

This option will be covered in the next section on dialog applications since it has no relevance to frames.

We will leave this setting at its default value.

Min Button, Max Button, Close Button

I will consider these three options together since they are all related. They correspond to the flags `wxMINIMIZE_BOX`, `wxMAXIMIZE_BOX` and `wxCLOSE_BOX`. These flags determine if the frame displays these boxes or not as shown below.

All options selected



Min Button unselected



Max Button unselected



Close Button unselected



We will set the 'Max Button' option and leave the others as is.

Selected Profile

This drop down box offers you a choice of profile to use. This affects what compiler will be used to compile this project. At present you have a choice between Microsoft's compiler and the MingW port of GCC.

Alter the contents of the drop down box to reflect the compiler you want to use.

Now that we have looked at the options it is time to create the project.

Click on the [Create] button.

The dialog will close and the form editor will open.

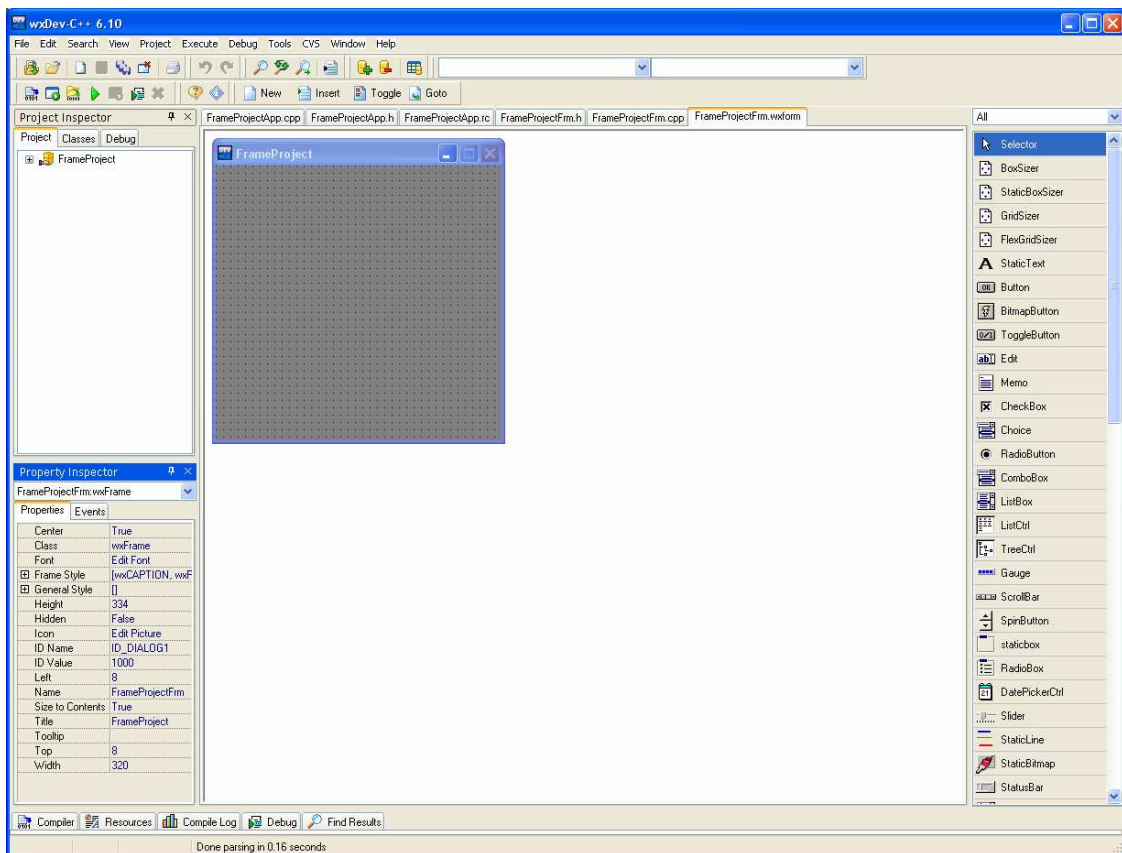


Figure 8.12 – The form designer showing a new frame

So now that we have a new frame what can we do with it? Well the first thing to try is compiling it.

Press <F9>.

After compiling the window will appear looking something like this.

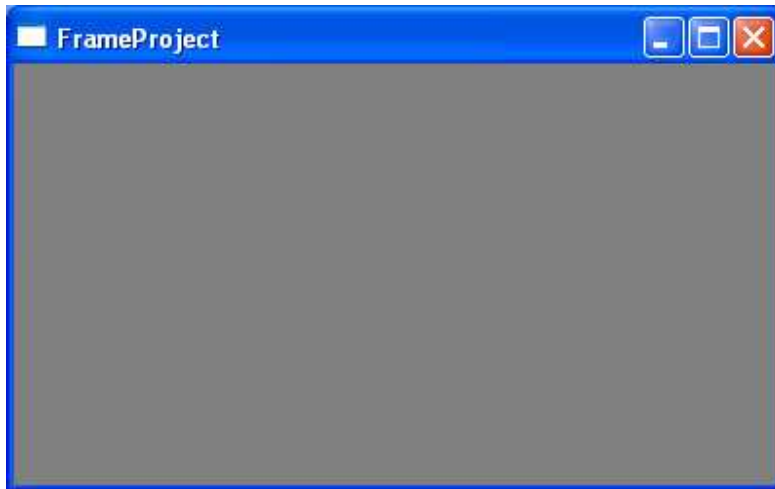


Figure 8.13 – The compiled version of our frame.

Wait a minute, that does not look like our frame in the designer, the width and height are all wrong. The reason for this is that the code to change the compiled frame is not generated until you make a change to the frame. We can try this by dragging the edge of the frame slightly then trying to compile again. This time the result should look like the frame in the designer.

There are two ways to alter the frame one is by direct manipulation; the other is by changing properties.

Direct manipulation of the frame

Direct manipulation of the frame is limited to dragging the frame by the title bar to a different location in the designer. This alters the Top and Left properties of the frame, which affects where the compiled frame appears on the screen. It is also possible to drag the borders of the frame to resize it. This affects the Height and Width properties of the frame.

Changing frame properties

The properties for the frame are found in the Property Inspector shown below.

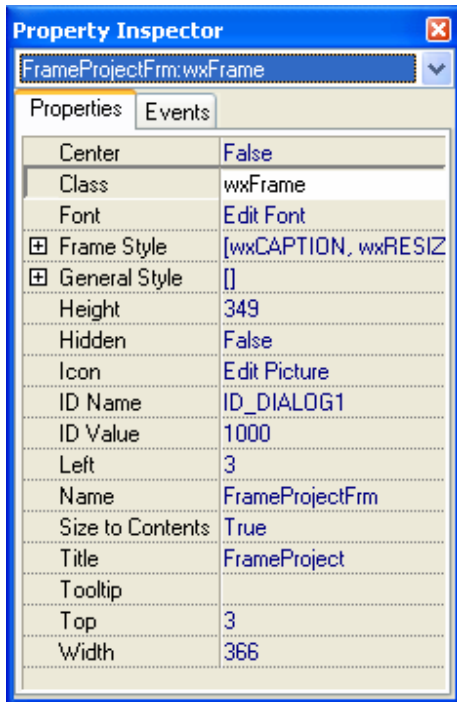


Figure 8.14 – The Property Inspector

If the Property Inspector is not showing the frame then click on the drop down box and select the option that ends in :wxFrame.



Figure 8.15 – Selecting the frame in the Property Inspector

Now let us consider the properties shown in Figure x.x one by one. We will alter some of these to create our example project.

Center

The property Center is a Boolean property. This means that it takes a true/false value. If Center is set to **true** the function Center() is called when the frame is created. This means the frame will be centered on the monitor when it is created. If it is set to **false** the frame will be created at the location on the monitor specified by the properties Top and Left.

We will set it to false.

Class

The property Class allows you to use custom classes. We will cover the use of this property later in the book in Chapter 13 - Creating and using other controls.

Font

This option sets the font that will be used by this frame and controls that are children of this frame.

We will leave this setting alone since we wont be adding any controls in this chapter.

Frame Style

A style in wxWidgets affects how a control looks and works. Some styles are specific to certain platforms others are more generic. A wxFrame has two sets of styles, the general styles applicable to wxWindow classes from which it derives and it's own unique styles. This section is devoted to the frame's own unique styles.

By default the Frame Style property is shown as a long line of flags. This property can be expanded by clicking on the '+' just before the caption Frame Style. A full list of the styles will then be shown.

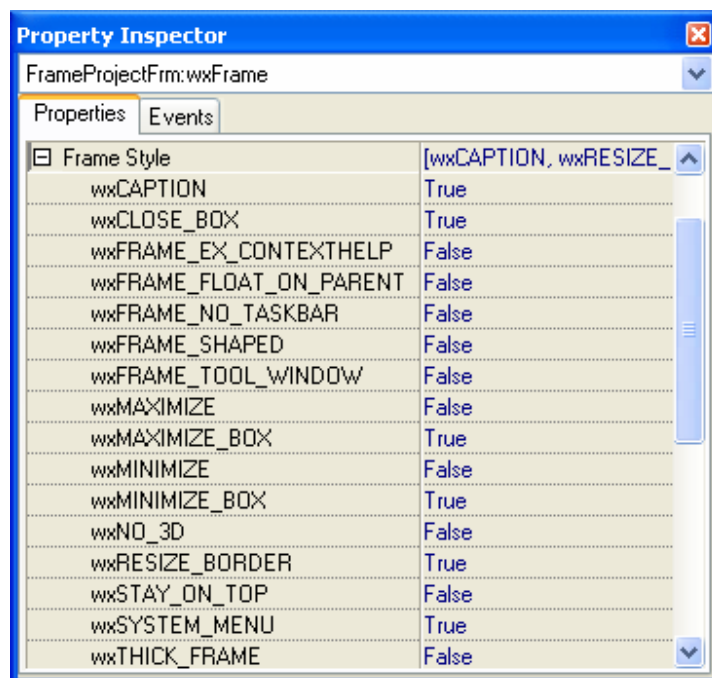


Figure 8.16 – The expanded list of frame styles

Some of these styles we have already met when we created the frame. They are all Boolean values, that is true or false. If the property is set to true the flag will be

added to the list of flags passed to the frame when it is created at runtime. We will now consider each style in turn.

Flag Name	Flag behaviour or style
wxCAPTION	We have already come across this flag when we created the frame. Remember setting this to false will remove not only the caption, but also the border from your frame.
wxCLOSE_BOX	This is another property we have already seen it enables or disables the close box on the frame.
wxFRAME_EX_CONTEXTHELP	This flag is supposed to add a query button to the frames caption bar. However the way it is currently implemented in wxDev-C++ means that it does nothing.
wxFRAME_FLOAT_ON_PARENT	This cause the frame to float above its parent frame which means it will always appear on top of its parent.
wxFRAME_NO_TASKBAR	This flag causes a frame to be created which does not show up in the taskbar. On the Windows platform the frame will be minimized to the corner of the desktop.
wxFRAME_SHAPED	This flag creates a frame which may be displayed with areas cut out such as some splash screens.



Figure 8.17 – The wx shaped frame sample

wxFRAME_TOOL_WINDOW

This creates a frame with a smaller basic caption area and close button. This type of frame is often used for tool box style windows.



Figure 8.18 – A tool window frame

wxMAXIMIZE

This flag causes the frame to be displayed in a maximized state.

wxMAXIMIZE_BOX

This is another property we have already seen it enables or disables the maximize

<code>wxMINIMIZE</code>	box on the frame. This flag causes the frame to be displayed in a minimized state.
<code>wxMINIMIZE_BOX</code>	This is another property we have already seen it enables or disables the minimize box on the frame.
<code>wxNO_3D</code>	This flag overrides the native 3D drawing effect for this frames child components.
<code>wxRESIZE_BORDER</code>	We have already seen this flag when we were creating the frame. It creates a frame that the user can resize.
<code>wxSTAY_ON_TOP</code>	This is another flag we have come across. It causes this frame to stay on top of all other windows. You can see this operating with Windows Task Manager.
<code>wxSYSTEM_MENU</code>	This is another flag we have seen which removes the icon in the corner of the frame as well as the menu attached to it. This also has the side effect of removing the minimize, maximize and close buttons.
<code>wxTHICK_FRAME</code>	We also encountered this flag earlier. As mentioned before it has no result and is the same as setting <code>wxRESIZE_BORDER</code> .

General Style

The general style property contains the styles that apply to all wxWindows controls. Since wxFrame derives from this class it also has the ability to apply these styles.

To access them click on the little cross '+' besides the property 'General Style' in the Property Inspector. The complete list will drop down as shown below.

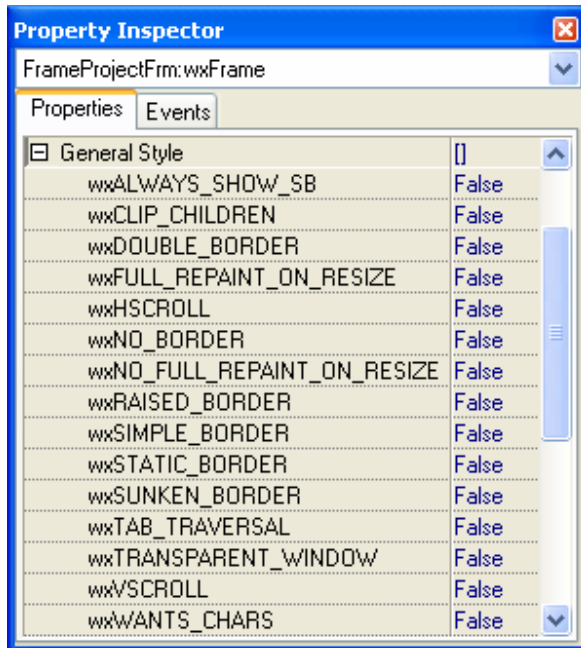


Figure 8.19 – The full list of General Style properties

All these styles are Boolean values. If the property is set to true the flag will be added to the list of flags passed to the frame when it is created at runtime. We will now consider each style in turn.

Flag Name	Flag behaviour or style
wxALWAYS_SHOW_SB	If window has scrollbars then this flag causes them to be permanently displayed even when they are not needed.
wxCLIP_CHILDREN	This flag removes flicker that is caused when child controls are painted.
wxDOUBLE_BORDER	This has no effect on frames.
wxFULL_REPAINT_ON_RESIZE	This flag causes the whole frame to be repainted when it is resized. This flag only needs to be used when there are redrawing problems.
wxHSCROLL	Causes the window to display a horizontal scrollbar.
wxNO_BORDER	This flag does not alter the frame at



Figure 8.20 – Frame displaying a horizontal scrollbar.

<p>wxNO_FULL_REPAINT_ON_RESIZE</p> <p>wxRAISED_BORDER</p> <p>wxSIMPLE_BORDER</p> <p>wxSTATIC_BORDER</p> <p>wxSUNKEN_BORDER</p> <p>wxTAB_TRAVERSAL</p> <p>wxTRANSPARENT_WINDOW</p>	<p>all.</p> <p>This flag is used by default and disables fully repainting the window when it is resized.</p> <p>This has no effect on frames.</p> <p>This has no effect on frames.</p> <p>This has no effect on frames.</p> <p>This has no effect on frames.</p> <p>This flag enables tab traversal.</p> <p>This flag stops the window responding to paint events. It is usually used for child controls which need to appear transparent. Used on frames it can cause all or part of the frame not to be drawn.</p>
---	--

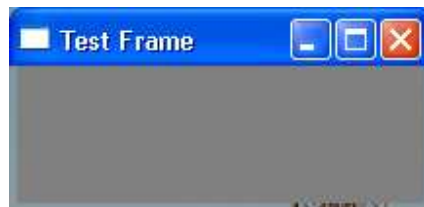


Figure 8.21 – A frame whose border has not been fully drawn.

wxVSCROLL

Causes the window to display a vertical scrollbar.



Figure 8.22 – Frame displaying a vertical scrollbar

wxWANTS_CHARS

This flag indicates that the window should generate key events for all keys, even keys like TAB which are normally used for navigation.

Height

This property is the overall height of the frame in pixels.

We will set this to 100.

Hidden

This property is currently unused and can be ignored.

Icon

This property allows us to choose an icon that is displayed on the frame. wxDev-C++ has a small library of icons that can be used for this purpose. To add an icon you need to click on the button on the right hand side of the property label ‘...’



Figure 8.23 – Altering the icon property

This will open a dialog like the following

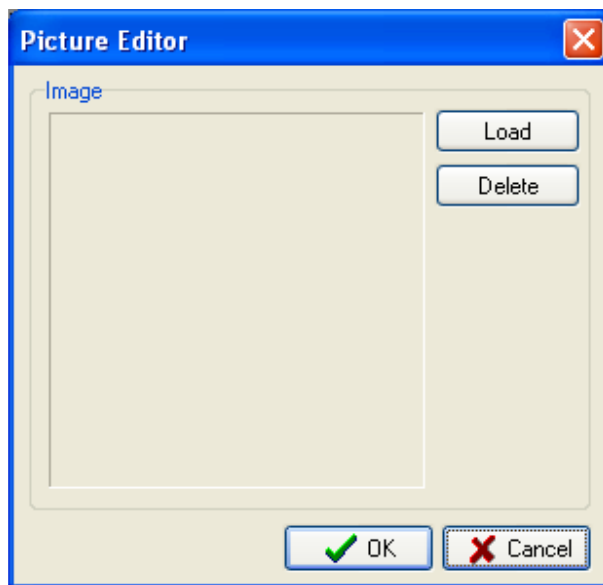


Figure 8.24 – The load picture dialog

Click on the load button and in the dialog that opens navigate to the folder where wxDev-C++ is installed. There should be a folder there named 'Icons'. Open this folder.

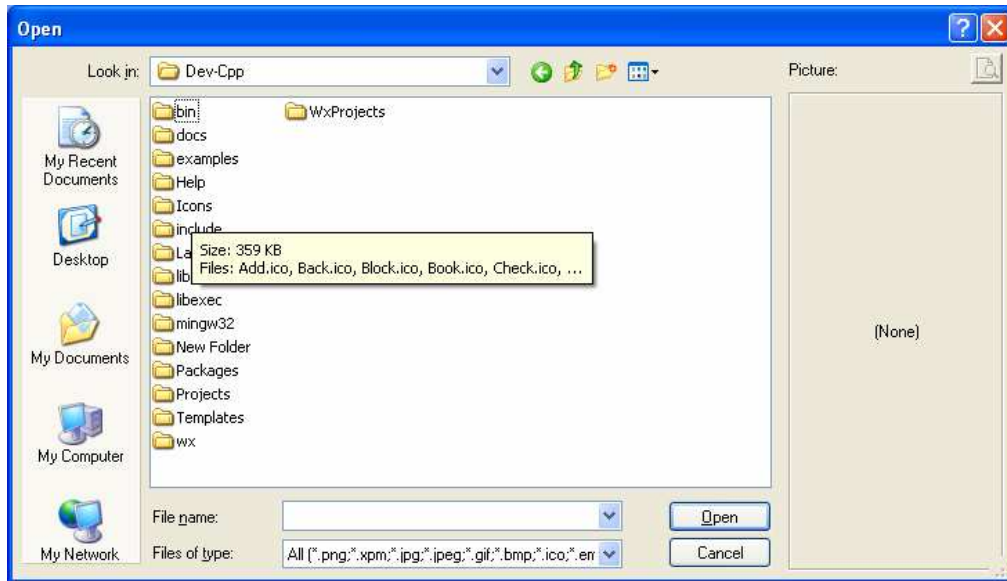


Figure 8.25 – The file chooser

We will choose the Ufo.ico for this sample.

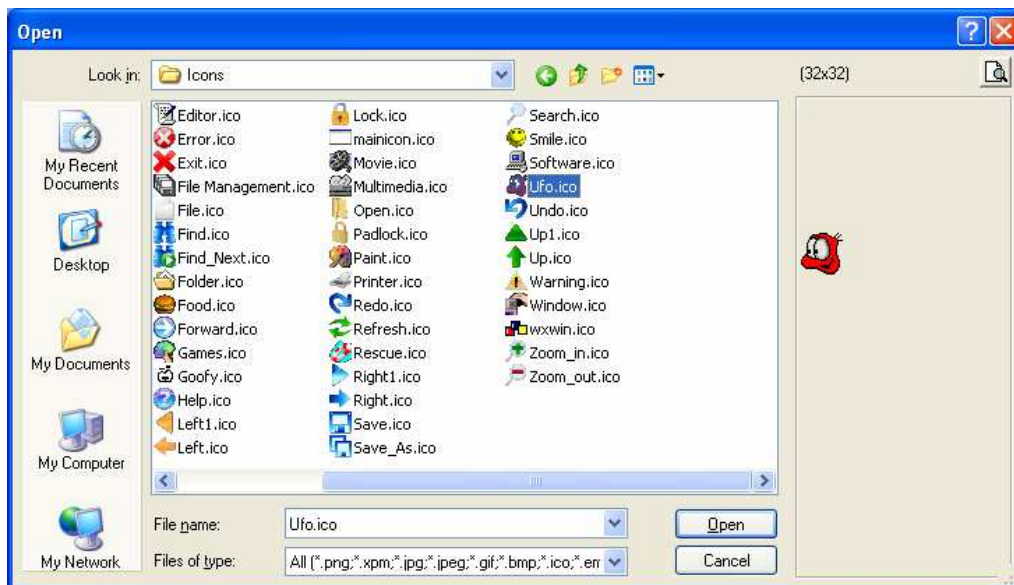


Figure 8.26 – The ufo icon

Select Ufo.ico
 Click [Open]
 Then on the next dialog click [OK]

You will notice that the icon displayed on the designer has not changed. However if we compile the program we will see that the icon has been altered.

Press <F9> to compile and run.



Figure 8.27 – The compiled frame showing our new icon

ID_Name

This property assigns a name to a constant value used to identify the frame. This name is used by the event handling mechanism. It is often best to just accept the pre-generated name.

We will leave this value alone.

ID_Value

This property is the value associated with the ID_Name. As mentioned the ID_Name is a constant that is equal to this value. Again it is often best to leave this alone.

We will leave this value alone.

Left

This property sets where the left edge of the frame will be placed in relation to the left hand side of the monitor screen in pixels.

We will set this to 200

Name

This is the name used you used on the frame creation dialog. It is used as the class name of your class which derives from wxFrame. It is generally better to leave this alone, especially once you start writing code since the compiler uses this name to reference all the code related to the frame. In fact if you do try to change it you will receive the following warning.

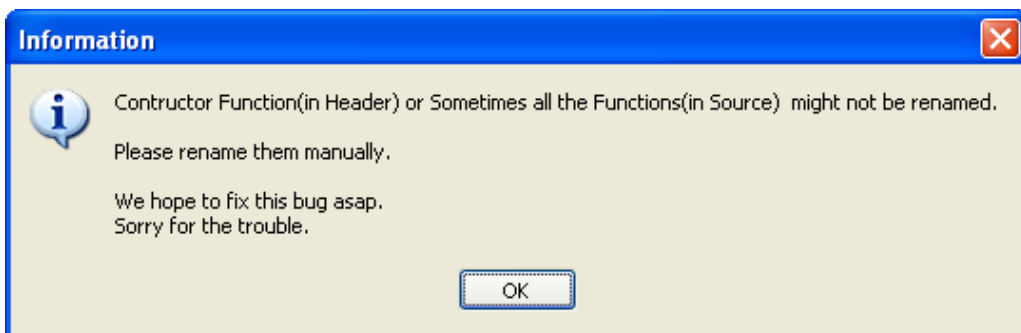


Figure 8.28 – Warning that changing the component name might cause problems

We will leave this value alone.

Size to contents

This property is used when a sizer is added to the frame. It causes the following line of code to be generated.

```
GetSizer()->SetSizeHints(this);
```

This tells the sizer to set the minimal size of the frame to its own minimal size. Then resize the frame to this size.

Title

This is a string property that corresponds to the caption displayed on the frame.

We will change this to 'Hello World'



Figure 8.29 – The frame's caption

Tooltip

This sets a string property that is shown in a little box when the mouse pointer hovers over the frame.

We will set this to 'I Love wxWidgets'



Figure 8.30 – The tooltip displayed on the compiled frame

Top

This property sets where the top of the frame will be placed in relation to the top of the monitor screen in pixels.

We will set this to 200

Width

This property is the overall width of the frame in pixels.

We will set this to 200.

We will now see the completed frame in action

Now press <F9> to compile and run.



Figure 8.31 – The completed frame project

Dialog Project

We are now going to create the wxDialog equivalent to the previous project. To do so

Go to File|New|Project...

You will get the same new project dialog as before.

- Select wxWidgets Dialog
- Change the project name to DialogProject
- Make sure the radio button next to 'C++ Project' is selected
- Click [OK]

Next you will be prompted for a location to save this project to.

- Create a new folder in your 'Project' folder
- Name the new folder DialogProject
- Enter the DialogProject folder
- Click [Save]

Next you receive a similar project options dialog as we received for the frame project

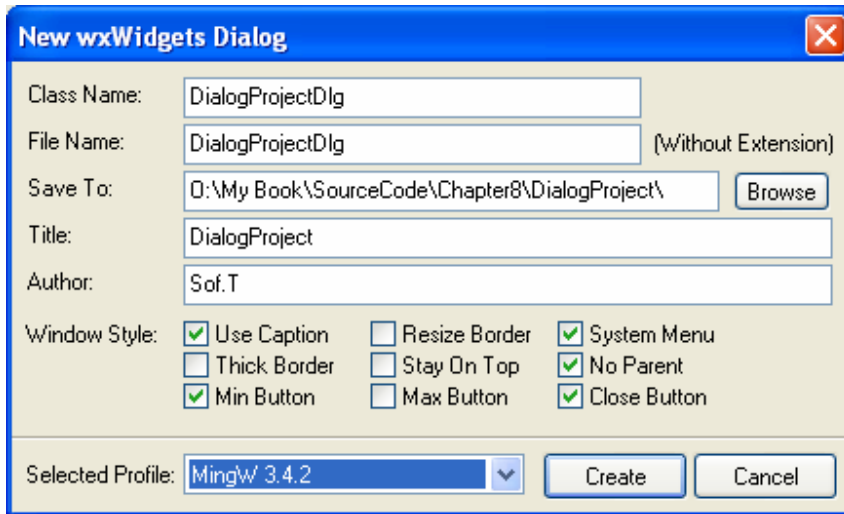


Figure 8.32 – The wxDialog project settings dialog

There are few differences to the frame project dialog. The caption has changed to New wxWidgets Dialog. The automatically generated Class and File names now end in ‘Dlg’ rather than ‘Frm’ and that is about it.

One point we need to cover here is the ‘No Parent’ option. This did nothing when generating a frame project. However it is necessary for dialog applications. This option corresponds to the flag `wxDIALOG_NO_PARENT`. It tells the application that this dialog does not have a frame as a parent.

Leave all the settings as they are
Click the [Create] button

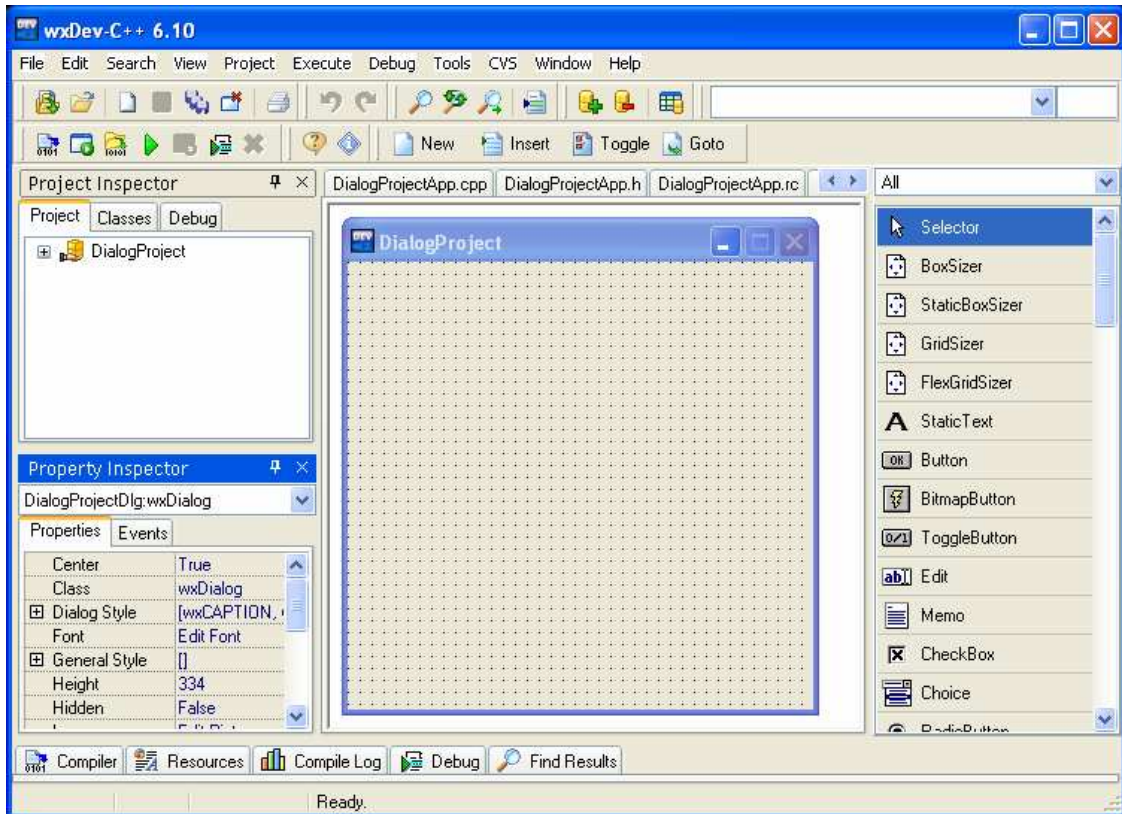


Figure 8.33 – The new dialog project

Once the new project has been fully created you may notice one particular change to the wxFrame project. The wxDialog is a light grey colour rather than the wxFrame's dark grey colour. This is not a mistake. The designer is mimicking the visual difference between a frame and a dialog in wxWidgets.

A dialog can be altered in exactly the same manner as a frame. If you look at the list of options in the Property Inspector you will notice that they are almost exactly the same.

The only differences can be found in the list of flags found under 'Dialog Style'.

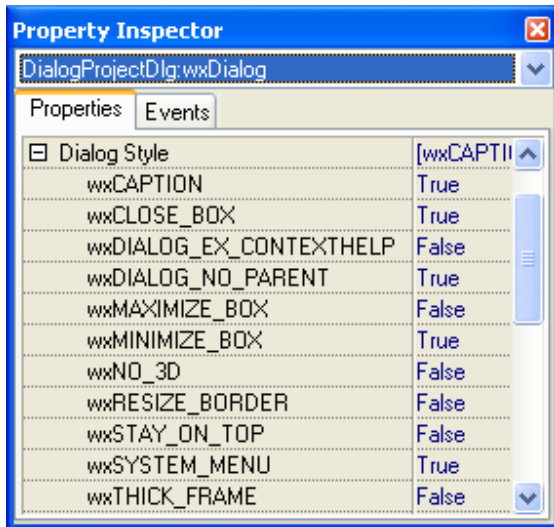


Figure 8.34 – The dialog style flags

Dialog style flags

Flag Name	Flag behaviour or style
wxCAPTION	Same meaning as for frame
wxCLOSE_BOX	Same meaning as for frame
wxDIALOG_EX_CONTEXTHELP	The same comments apply here as for wxFRAME_EX_CONTEXTHELP
wxDIALOG_NO_PARENT	This tells the application that this dialog does not have a parent window.
wxMAXIMIZE_BOX	Same meaning as for frame
wxMINIMIZE_BOX	Same meaning as for frame
wxNO_3D	Same meaning as for frame
wxRESIZE_BORDER	Same meaning as for frame
wxSTAY_ON_TOP	Same meaning as for frame
wxSYSTEM_MENU	Same meaning as for frame
wxTHICK_FRAME	Same meaning as for frame

Project Settings

To complete our dialog project we will alter the following settings

- Height – change to 100
- Icon – change icon to ‘Food.ico’
- Title – change to “Dialog Hello World”
- ToolTip – change to “wxDev-C++ rocks”
- Width – change to 230

Press <F9> to compile and run



Figure 8.35 – The completed dialog project

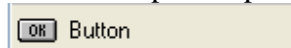
Multiple Frame Project

In this and the following section we are going to look at how to add more frames and dialogs to an application and how to use them. To begin create a new 'wxWidgets' Frame project.

- Open wxDev-C++
- Select File|New|Project...
- Select wxWidgets Frame
- Change Project Name to "MultipleFrames"
- Make sure C++ Project is selected
- Click the [OK] button
- Create a new folder in your projects
- Call the folder "MultipleFrames"
- Save the project in this folder
- On the next dialog click the [Create] button

We are now ready to begin this project. The intention is to add a new frame and a new dialog to this project and demonstrate the code needed to display them. First we need somewhere to put the code to open the second frame and second dialog. To do this we will add two buttons to the frame and use one to open new frames, the other to open new dialogs.

On the component palette find the 'Button' component



Select the button component and then click on the frame

Repeat this to create a second button

You should now have a frame that looks something like this

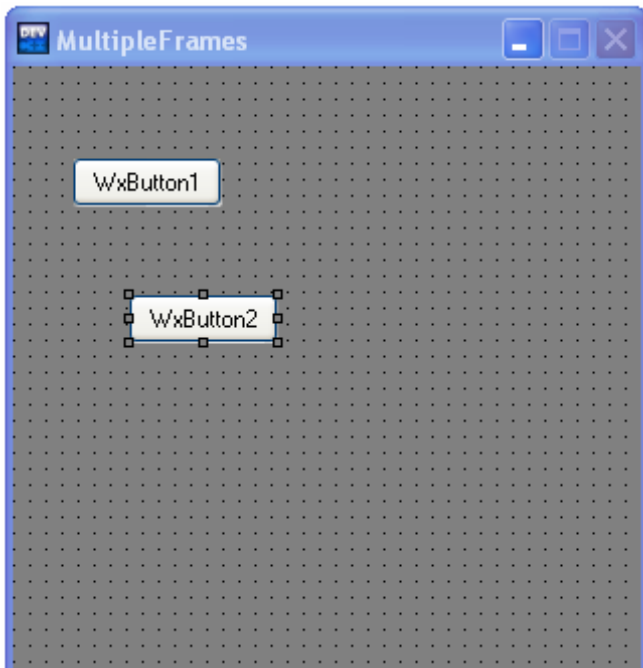


Figure 8.36 – The frame with two button controls added

Alter the two buttons as follows

Button titled WxButton1

- Change the Left property to the value 3
- Change the Top property to the value 3
- Change the Width property to the value 100
- Change the Label property to the value “Open New &Frame”
- Select the Events tab in the Property Inspector

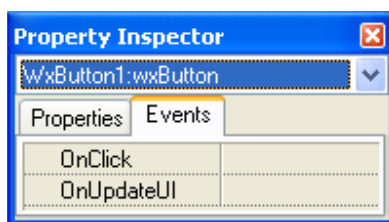


Figure 8.37 – The events tab

In the drop down box next to OnClick select <Add New Function>

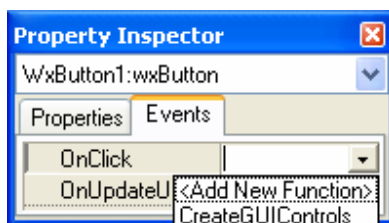


Figure 8.38 – The new function option

Return to the form designer by clicking on the tab labelled MultipleFramesFrm.wxform



Figure 8.39 – The form designer page

Now to create our second button

Button titled WxButton2

- Change the Left property to the value 3
- Change the Top property to the value 32
- Change the Width property to the value 100
- Change the Label property to the value “Open New &Dialog”
- Select the Events tab in the Property Inspector
- In the drop down box next to OnClick select <Add New Function>
- Return to the form designer

You should now have something resembling this

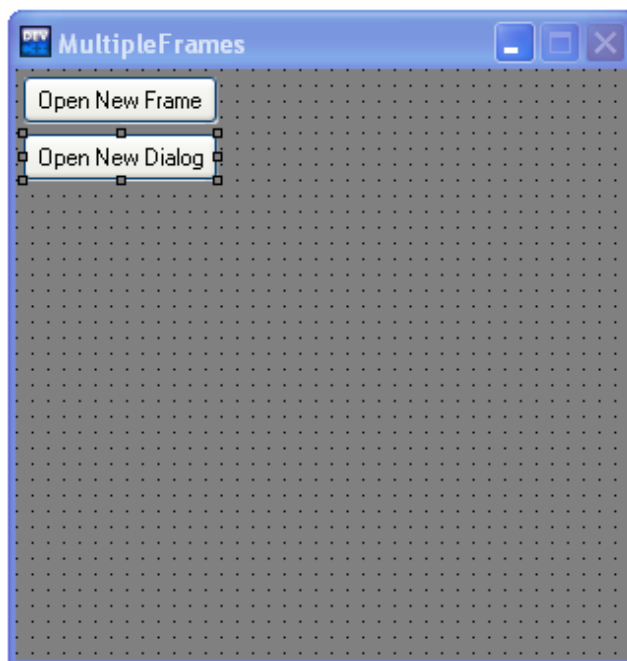


Figure 8.40 – The frame

Now to alter the frame

- Change the With property to 200

Change the Height property to 100

Finally you end up with this



Figure 8.41 – The completed main application frame

Now we will move on to adding another frame and dialog to this application.

Adding a New Frame

The first thing we need to do is add a new frame

Goto the menu option File|New|New wxFrame
Click the [Yes] button on the following dialog

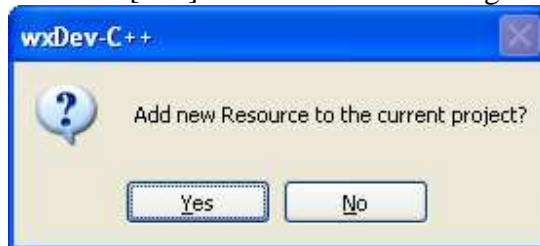


Figure 8.42 – Add new frame to the existing project dialog

You will recognize the next dialog

Change the Class Name to “SecondaryFrm”
Change the File Name to “SecondaryFrm”
Change the Title to “Secondary Frame”
Click the [Create] button

On the newly generated frame alter the following properties

Change Height to 100
Change Width to 250
Set the wxFRAME_NO_TASKBAR flag to True

Add a new button to the frame and set the button properties to the following

Change Left to 84
Change Top to 23
Change Label to “&Close”

On the buttons Event Tab create a new OnClick event

In the newly generated code add the following line

```
Destroy();
```

The Destroy() function comes from the base wxWindow class. Its purpose is to safely close the window while dealing with any unprocessed events.

Now we need to see how to use this frame in our existing project.

In the source code file "MultipleFramesFrm.cpp" look for the lines

```
//Do not add custom headers
//wxDev-C++ designer will remove them
////Header Include Start
////Header Include End
```

This is the section where wxDev-C++ adds any header files it needs to. After the line
////Header Include End

```
Add the line #include "SecondaryFrm.h"
```

This adds the declaration of our SecondaryFrm class to the file MultipleFramesFrm.cpp. This allows us to use the class SecondaryFrm within the code in the file MultipleFramesFrm.cpp.

Now scroll down through the code in MultipleFramesFrm.cpp until you reach these lines.

```
/*
 * WxButton1Click
 */
void MultipleFramesFrm::WxButton1Click(wxCommandEvent&
event)
{
    // insert your code here
}
```

Now add the following lines within this block of code

```
void MultipleFramesFrm::WxButton1Click(wxCommandEvent&
event)
{
    // Create a new frame
    SecondaryFrm * TempFrame = new SecondaryFrm(this);
    TempFrame->Show();
}
```

```
}
```

The line `SecondaryFrm * TempFrame = new SecondaryFrm(this);` creates an instance of our class `SecondaryFrm` and assigns it to a temporary pointer of the type `SecondaryFrm`. We provide the argument `this` in the constructor which makes this instance of the class `MultipleFramesFrm` class the parent. The benefit of this is that when you close or destroy a parent class all its children will also be destroyed. So when you want to exit an application you only want to close the main window not every window.

The second line `TempFrame->Show();` displays the newly created frame.

To see what we have so far

Press <F9> to compile and run the application
Then click on the [Open New Frame] button

The following window should be displayed



Figure 8.43 – Our secondary frame

The appearance of this frame may surprise you. What you will notice is that the button fills the whole of the frame. This is not an error on wxDev-C++'s part. Rather this is a feature of wxWidgets. When a frame contains only one control it is enlarged to cover the whole of the frames client area.

You can experiment with the following features

Try using the close button to close a Secondary Frame.
Try creating several Secondary Frames. (You will need to move them out of the way)
Try closing the MultipleFrame frame.

You will notice that the new secondary frames are not shown on the taskbar. You can create several instances of the same class. All instances are closed when you close the parent frame.

Now we will continue this example by adding a dialog.

Adding a New Dialog

We add a new dialog in the same manner as adding a new frame.

- Goto File|New|New wxDialog
- Click the [Yes] button
- Change the Class Name to “SampleDlg”
- Change the File Name to “SampleDlg”
- Change the Title to “Sample Dialog”
- Uncheck the No Parent option
- Check the Stay On Top option
- Click the [Create] button

Add two new buttons to the dialog. Then alter the dialog and button properties as follows.

SampleDlg

- Change Height to 80
- Change Width to 200

WxButton1

- Change ID Name to wxID_OK
- Change Label to “&OK”
- Change Left to 12
- Change Top to 14

WxButton2

- Change ID Name to wxID_CANCEL
- Change Label to “&Cancel”
- Change Left to 102
- Change Top to 14

Now in the file “MultipleFrames.cpp” look for the include file we added earlier (Line16) and add these two lines under it.

```
#include "SampleDlg.h"  
#include <wx/msgdlg.h>
```

The first line allows us to use our new dialog, the second line allows us to use messageboxes.

Then look for this function

```
/*  
 * WxButton2Click  
 */
```

```
void MultipleFramesFrm::WxButton2Click(wxCommandEvent&
event)
{
    // insert your code here
}
```

Alter this function by adding this code within it.

```
// insert your code here
SampleDlg TempDlg(this);
if(TempDlg.ShowModal() == wxID_OK)
{
    wxMessageBox("User Pressed OK");
}
```

Before we enter into any explanations

Press <F9> to compile and run the program.
Try experimenting with the [Open New Dialog] button and dialogs



Figure 8.44 – The compiled Sample Dialog

You should have discovered some interesting results. When a dialog is displayed you can no longer interact with the “Multiple Frames” frame. This is due to calling it with the `ShowModal()` function. This disables all parts of the program except the dialog until the dialog is closed.

You will also find that if you press the [OK] or [Cancel] buttons the dialog closes. But wait a minute we didn’t add any code to the buttons to make this happen. What we did do is change the button’s ID Name. With dialogs there are certain preset ID Name values that trigger functions built into the dialog `wxID_OK` and `wxID_CANCEL` are two of these.

Finally you will notice that when we close the dialog via the [OK] button a message box pops up to tell us that we pressed OK. This is the result of these lines.

```
if(TempDlg.ShowModal() == wxID_OK)
{
    wxMessageBox("User Pressed OK");
}
```

When the button causes the dialog to close the function `ShowModal()` returns the ID Name value of the button. We check for the value `wxID_OK` being returned and if we receive it we show a message box since we know only the OK button could return this value.

There is much more to learn about `wxFrame` and `wxDialog` and as ever the `wxWidget` help documentation is excellent. It is also worth looking at `wxTopLevelWindow` and `wxWindow` since they both derive from this. Now that we have learnt a little about frames and dialogs let us begin to create a sample application.

Sample Application Part 1 – The outline

Starting with this chapter we are going to design and build an application. This will be based on the previous sample program the HTML editor. However this will add many features that will make it a useful program. A screenshot of the completed program is shown below.

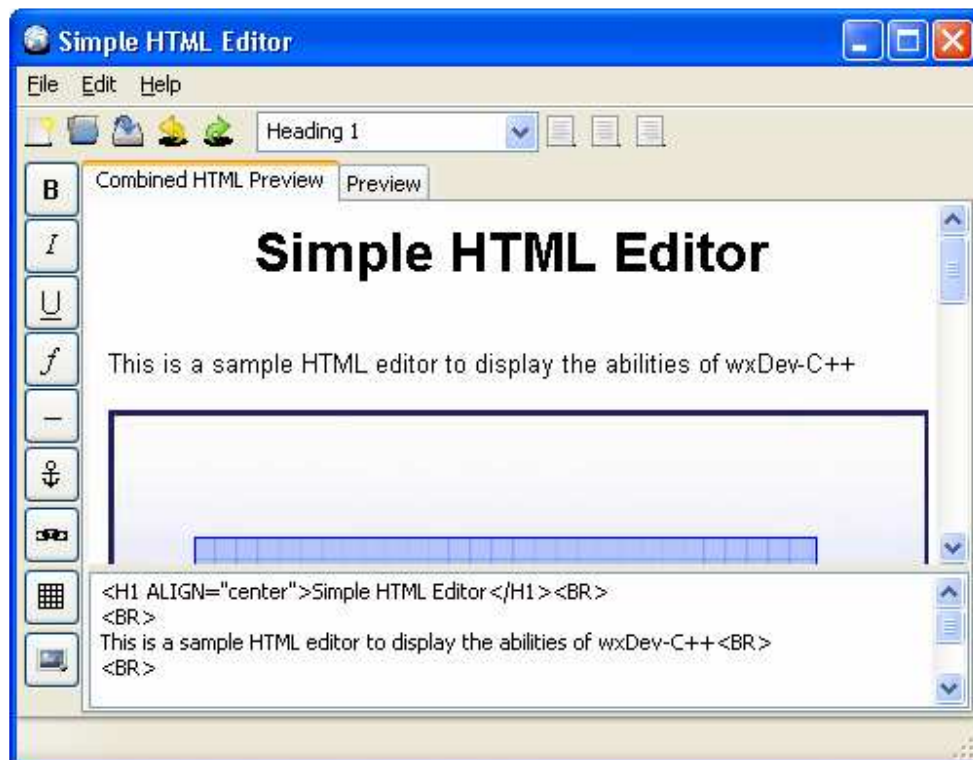


Figure 8.45 – The completed HTML editor.

In this chapter we will create the frames and dialogs we will need for the program. In chapter 10 we will look at the various components `wxDev-C++` makes available for us to use. We will end the chapter by adding the various components needed to make this program look like the one in the figure. In chapter 11 we will look at how and where to add code to make it all work. Then in chapter 12 we will consider various points on how to make it a polished professional program.

Before we start you will need to download and install the TangoImages.devpak from the [Bonus download section](#) of the sourceforge project site.

Without anymore ado let us start creating the frames. We are doing to create 1 extra frame and 4 dialogs. Instead of holding your hand all the way through I am going to tell you what creation and design time options you will need to set for each frame/dialog. If you run into trouble creating these the completed frames and dialogs are contained in the downloadable source code that accompanies the book. They are located in the Chapter 8 folder.

Start a new wxFrame application.
Set the project name to 'HTMLEdit'
Save the project in a new folder called 'HTMLEdit'
Set the frame properties according to the following table

Creation Properties

Class Name	HTMLEditfrm
File Name	HTMLEditFrm
Title	Simple HTML Editor
Author	Put your name here
Window Style	Use Caption, Resize Border, System Menu, No Parent, Min Button, Max Button, Close Button

Design time properties

Height	402
Width	553
Icon	Use internet-web-browser16x16.png from the tango icons installed from the TangoImages devpak into Dev-Cpp/Images

For this application we will have a splash screen. We will use a standard frame for this purpose.

Create a new wxFrame
Select yes to add to current project
Then set the properties according to the following table

Creation Properties

Class Name	Splashfrm
File Name	SplashFrm
Title	Splash Form
Author	Put your name here
Window Style	Stay On Top, No Parent

Design time properties

Height	164
Width	220
Frame Style	wxSTAY_ON_TOP, wxDIALOG_NO_PARENT, wxFRAME_NO_TASKBAR, wxFRAME_SHAPED

Now we need to create our first dialog. We will be using this as an about box.

Create a new wxDialog

Select yes to add to current project

Then set the properties according to the following table

Creation Properties	
Class Name	AboutBoxDlg
File Name	AboutBoxDlg
Title	About Simple HTML
Author	Put your name here
Window Style	Use Caption, System Menu, No Parent, Close Button
Design time properties	
Height	284
Width	265

The second dialog will be used to create tables in HTML.

Create a new wxDialog

Select yes to add to current project

Then set the properties according to the following table

Creation Properties	
Class Name	CreateTableDlg
File Name	CreateTableDlg
Title	Create Table
Author	Put your name here
Window Style	Use Caption, System Menu, No Parent, Close Button, Minimize Button
Design time properties	
Height	308
Width	236
Icon	Use internet-web-browser16x16.png from the tango icons installed from the TangoImages devpak into Dev-Cpp/Images

Our next dialog will be used to insert images into the HTML.

Create a new wxDialog

Select yes to add to current project

Then set the properties according to the following table

Creation Properties

Class Name	InsertImageDlg
File Name	InsertImageDlg
Title	Insert Image
Author	Put your name here
Window Style	Use Caption, System Menu, No Parent, Close Button, Minimize Button

Design time properties

Height	224
Width	238
Icon	Use internet-web-browser16x16.png from the tango icons installed from the TangoImages devpak into Dev-Cpp/Images

Our final dialog will be used for adding hyperlinks into the HTML document.

Create a new wxDialog

Select yes to add to current project

Then set the properties according to the following table

Creation Properties

Class Name	InsertHyperlinkDlg
File Name	InsertHyperlinkDlg
Title	Insert Hyperlink
Author	Put your name here
Window Style	Use Caption, System Menu, No Parent, Close Button, Minimize Button

Design time properties

Height	154
Width	373
Icon	Use applications-internet16x16.png from the tango icons installed from the TangoImages devpak into Dev-Cpp/Images

That concludes this chapter if you wish you can press <F9> to compile and run the program, but be warned it will look very boring at this stage. Make sure you save the project before closing it. In the next chapter we will start adding controls. By the end of the main chapter you will still only see the main frame, but it will be a whole lot more exciting.

Chapter 9 – The Component Palette

Introduction

The component palette is divided into several sections Sizers, Controls, Window, Toolbar, Menu, Dialogs, System, MMedia, Unofficial and All. 'All' unsurprisingly contains all the available controls. The other options provide a means of making it easier to locate specific controls.

In this chapter we will consider each of the palettes in turn and discuss the controls they contain. The chapter will end by adding to the sample application we began building in the last chapter.

Adding Components to a Frame or Dialog

The components are displayed in the palette on the right hand side of the IDE.

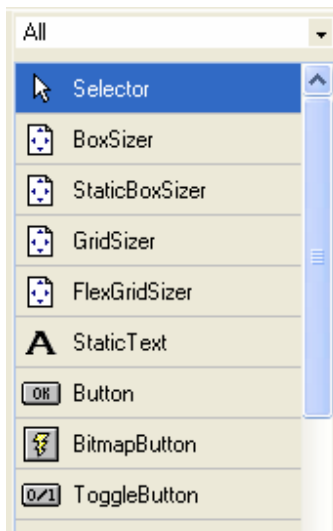


Figure 9.1 – The component palette

To place a component on a frame you first need to select it from the palette by clicking on it. It will be highlighted blue to show it is active.

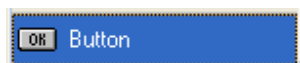


Figure 9.2 – A selected component

Next click on the designer form to drop the component.

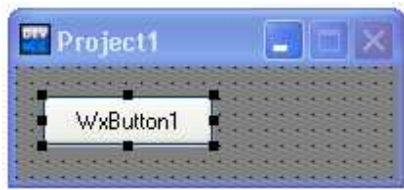


Figure 9.3 – The selected component added to the form

Some components have the ability to contain other components. The panel is an example of this. To add a component into a container component you first need to select the container. Do this by clicking on it, it will be highlighted with 8 small rectangles. Then select the component you wish to add from the palette. Now click on the container to place this component inside.



Figure 9.4 – A button placed within a panel

What if you select a component from the palette by mistake and want to change your mind? You can either select a different component or click on the selector option which appears at the top of every palette. This will deselect the component.

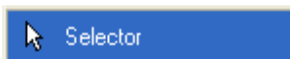


Figure 9.5 – The selector option

Altering Components

Just like frames there are two ways to alter components from within wxDev-C++, one is by direct manipulation; the other is by changing properties.

Direct Manipulation Of Components

To alter components via direct manipulation you need to select them. This is achieved by clicking over the component with the mouse. When a component is selected it shows 8 dark resizing squares on its edges.

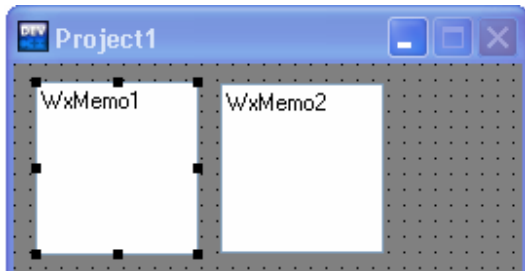


Figure 9.6 – Figure showing WxMemo1 as selected, WxMemo2 as unselected

If you hover over these resizing squares the mouse pointer will change once it does so you can press and hold the left mouse button. Moving the mouse will now resize the component. While resizing the component in this manner the resizing rectangles will be replaced with a red border and a small tooltip will appear to tell you the current width and height of the control.

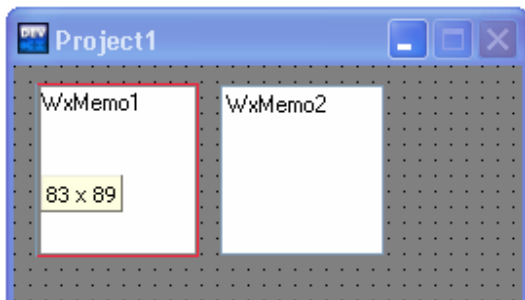


Figure 9.7 – Red resizing border and tooltip showing current width and height

It is possible to select more than one component at a time to do this hold down the shift key while clicking on each component you want to select. When more than one component is selected the 8 resizing rectangles are replaced with 4 light grey selection rectangles. These indicate that the selected controls can not be resized but can be moved. All the selected controls will be moved by the same amount.

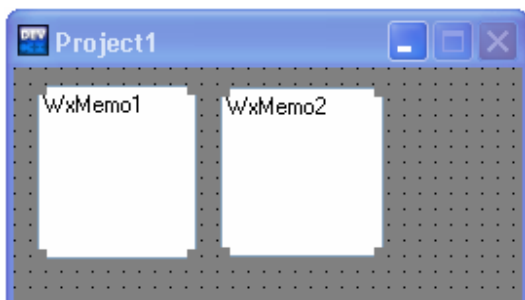


Figure 9.8 – Multiple selected components

To move a component press and hold the left mouse button while the pointer is over the component itself. You can then drag the component to a new position.

Alignment of controls can be tricky. Interfaces tend to look messy if the controls are all misaligned.

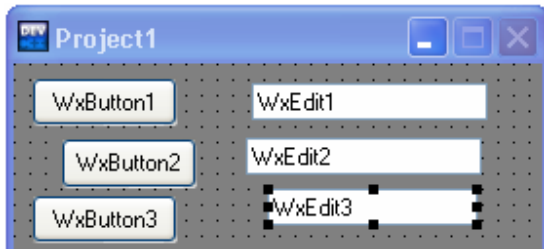


Figure 9.9 – Misaligned controls resulting in a messy interface

wxDev-C++ designer has a number of options to help you with aligning controls. These are based around the grid of little black dots you can see on the background of the design form. To make the most of this grid you need to bring up the context menu by right clicking on the frame designer.

The Designer Form Context Menu

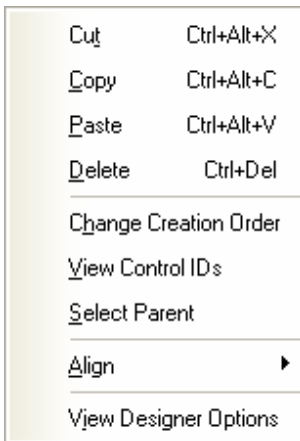


Figure 9.10 – The design form context menu

The options we will consider here are Align and View Designer Options. The first of these is the Align option. The hovering over the menu item produces a sub menu like the following.

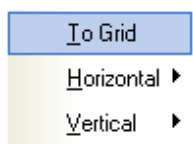


Figure 9.11 – Align menu

If you select the option To Grid all selected components will be moved to the nearest horizontal and vertical gridline. This has limited use. If you have several components selected then the other two options Horizontal and Vertical are of more use.



Figure 9.12 – The horizontal align menu

Choosing the option To Left will align all controls with the leftmost of the selected controls. The To Right will do the opposite and To Center will align all controls to the same center point.

The Vertical menu works in the same manner and has the options To Top, To Center and To Bottom.

The following picture is the result of aligning each pair of buttons and edit boxes with Align|Vertical|To Top. Then aligning all the buttons with Align|Horizontal|To Right and all edit boxes with Align|Horizontal|To Left.

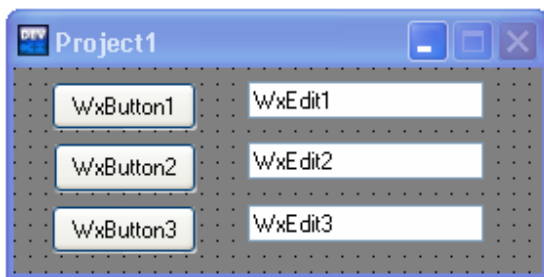


Figure 9.13 – A slightly neater interface after using the align options.

To customise the design grid choose the menu option View Designer Options. The following dialog will appear.

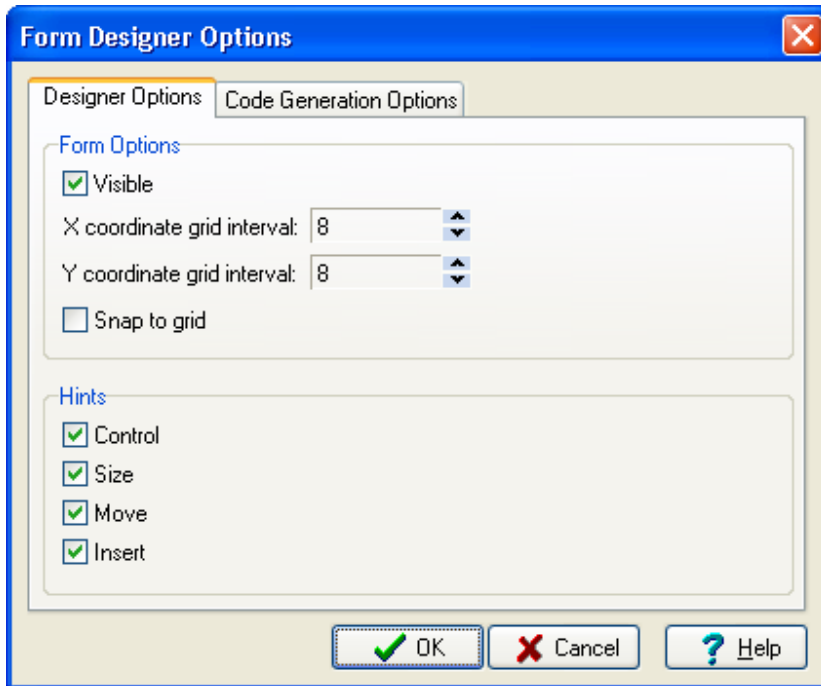


Figure 9.14 – The form designer options

The options available via this dialog allow you to alter the grid spacing, turn the grid on and off and enable snapping to the grid. The snapping option can be very useful and the controls will automatically jump to the nearest grid points and make it easier to align controls.

The Other Context Menu Options

The first four options are pretty obvious they cut, copy or delete the selected components, paste previously cut or copied components.

The Change Creation Order first brings up a warning dialog.



Figure 9.15 – Save before altering creation order warning

If you select yes then the following dialog is displayed

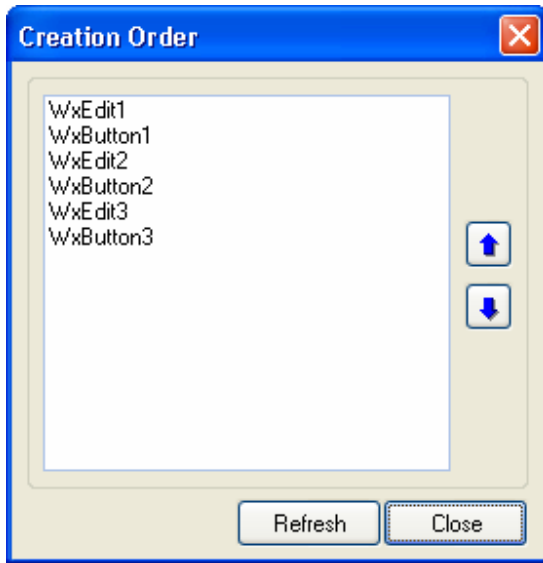


Figure 9.16 – Creation order dialog

This dialog allows you to alter the order your controls are created. If you check the `CreateGUIControls()` function in the corresponding `.cpp` file you will see that the controls are created in the same order as you arrange them in this dialog. This allows you to set the order in which the controls are traversed used the TAB key at runtime.

The option View Control IDs displays a dialog box which lists all the components on the form along with their ID number, ID name and Control Name.

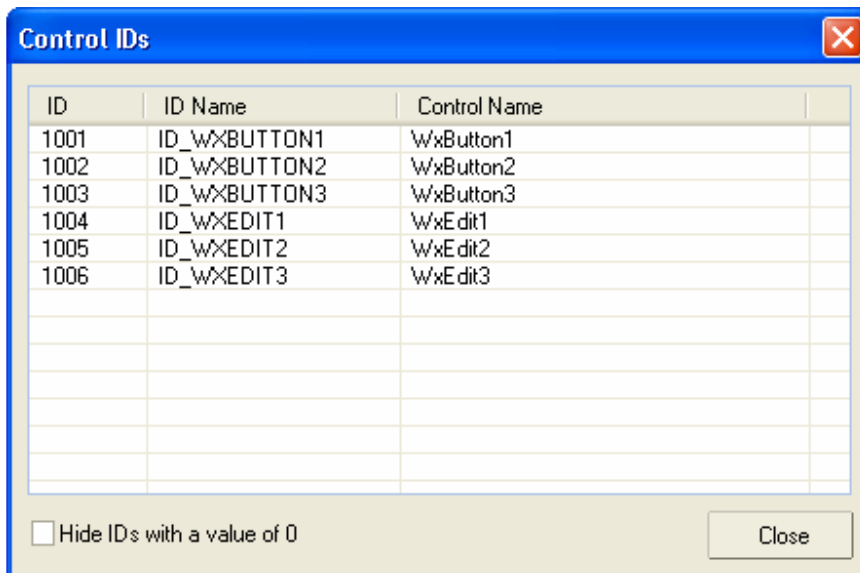


Figure 9.17 – List of controls and IDs

The next option is Select Parent, some components can only be contained by other controls, others such as `wxPanel` can act as a container. Occasionally you may wish to

change the parent or container object for a certain component. If you right click on it and choose Select Parent then you will get a list of possible parents.

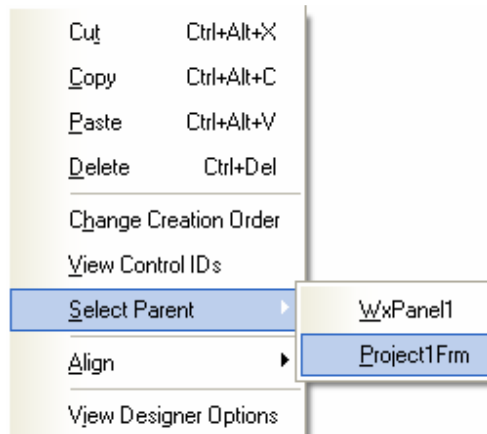


Figure 9.18 – Option of parents to choose from.

Component Properties

As discussed the other method of altering the components is by altering their properties. Some of these changes will show up in the designer, others will only be noticed once the application is compiled.

Before discussing the various components we will first look at a few properties that they all have in common. This means that we can concentrate on the unique properties of each component.

Common Properties

Alignment

This is a sizer related property which affects how the control will react to the sizer that contains it. At present this is designed as a single choice list. This will be covered under sizers.

Background color

This property alters the background colour of the control.

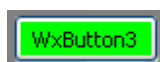


Figure 9.19 – Button with the background colour set to green

Base class

This is not a property as such. This setting allows you to use a different class name for the component allowing you to use your own wxWidgets derived

components. This is covered in more depth in Part 3. Don't alter this property unless you know what you are doing.

Border

This is a sizer related property which affects how much padding the control has between itself and the walls of the sizer that contains it. This will be covered under sizers.

Borders

This is a sizer related property which works with the Border option. This property allows you to specify which sides of the control will receive the padding. This will be covered under sizers.

Comments

This is a nice feature. Any text entered here is transferred into the source code as a comment that appears above the control at creation time. This can help you later when working on a big project to remember the purpose of this control.

```
/* This is the text entered as a comment
*/
WxButton1 = new wxButton(this, ID_WXBUTTON1,
```

Figure 9.20 – The code created by altering the comment property of a wxButton

Enabled

This property affects whether the control will be enabled or not at runtime. A disabled control is greyed out and doesn't respond to user input.

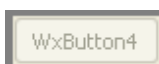


Figure 9.21 – A wxButton with the enabled property set to false.

Font

This option sets the font that will be used by this control and controls that are children of this control.



Figure 9.22 – A wxButton after the font property has been altered

Foreground Colour

This option sets the foreground colour of the control, this generally affects the font colouring.



Figure 9.23 – A wxButton after the foreground property has been altered to red

General Style

The general style property contains the styles that apply to all wxWindows controls.

To access them click on the little cross ‘+’ besides the property ‘General Style’ in the Property Inspector. The complete list will drop down as shown below.

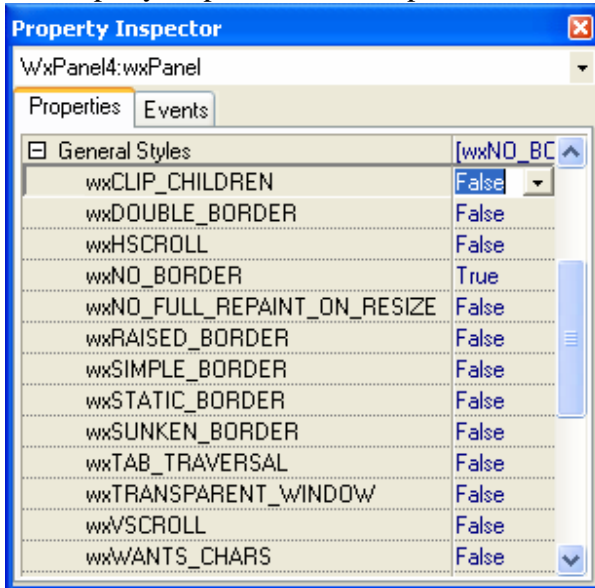


Figure 9.24 – The full list of General Style properties

All these styles are Boolean values. If the property is set to true the flag will be added to the list of flags passed to the frame when it is created at runtime. We will now consider each style in turn.

Flag Name

Flag behaviour or style

wxCLIP_CHILDREN

This flag removes flicker that is caused when child controls are painted.

wxDOUBLE_BORDER

Supposedly displays a double border on Windows and Mac platforms.



Figure 9.25 – wxPanel with double border set

wxHSCROLL

Causes the component to display a horizontal scrollbar.



Figure 9.26 – wxPanel displaying a horizontal scrollbar.

wxNO_BORDER

Causes the component to be drawn without a border.

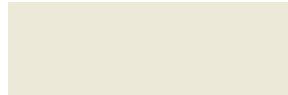


Figure 9.27 – wxPanel drawn without a border

wxNO_FULL_REPAINT_ON_RESIZE

This flag is used by default and disables fully repainting the component when it is resized.

wxRAISED_BORDER

Displays a raised border around the component.



Figure 9.28 – wxPanel drawn with a raised border

wxSIMPLE_BORDER

This displays a single line border around the component.



Figure 9.29 – wxPanel drawn with a simple border

wxSTATIC_BORDER

This displays a single line suitable for static components.



Figure 9.30 – wxPanel drawn with a static border

wxSUNKEN_BORDER

Displays a sunken frame around the component.



Figure 9.31 – wxStaticBitmap with sunken border

wxTAB_TRAVERSAL

This flag enables tab traversal on this component.

wxTRANSPARENT_WINDOW

This flag stops the component responding to paint events. It is usually used for child controls which

wxVSCROLL

need to appear transparent.
Causes the window to display a vertical scrollbar.

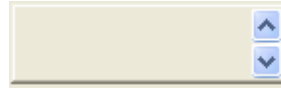


Figure 9.32 – Frame displaying a vertical scrollbar

wxWANTS_CHARS

This flag indicates that the window should generate key events for all keys, even keys like TAB which are normally used for navigation.

Height

This property sets to height of the control in pixels.



Figure 9.33 – A wxButton with the height property set to 62

Help Text

?

Hidden

This property affects whether the control will be displayed at runtime or not.



Figure 9.34 – A wxButton with the hidden property set to true

ID_Name

This property assigns a name to a constant value used to identify the frame. This name is used by the event handling mechanism. It is often best to just accept the pre-generated name. The exception is when you are using controls to carry out predefined actions. These actions are listed in the drop down box.

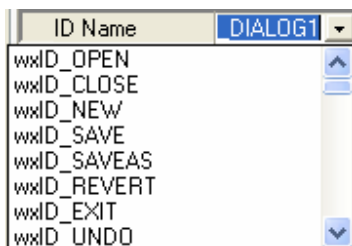


Figure 9.35 – The list of predefined ID_Names

ID_Value

This property is the value associated with the ID_Name. As mentioned the ID_Name is a constant that is equal to this value. Again it is often best to leave this alone.

Left

This sets the left position of the control in pixels in relation to its parent.

Name

This is the name you will use to refer to this instance of the component in your code. It is good to change this to a memorable and useful name, (See naming conventions in chapter 11), before you start coding. Once you have attached code to this control it makes it harder to alter.

Stretch Factor

This is a sizer related property that affects how much space a control is allowed to take up. This will be covered under sizers.

Tooltip

This property sets the text that will popup when the mouse hovers over the control

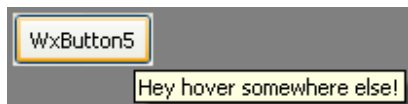


Figure 9.36 – The tooltip attached to a wxButton

Top

This sets the top position of the control in pixels in relation to its parent.

Width

This property sets the width of the control in pixels



Figure 9.37 – A wxButton with the width property set to 20

In addition to the above properties there are three properties that apply mainly to controls used to display values or accept values from the user. Two of these RHS and LHS Variables are strictly wxDev-C++ related, the third Validator Code is wxWidgets related. The purpose of these appears below.

LHS Variable

RHS Variable

Validator Code

wxWidgets uses validators for three different reasons. These are to fill or retrieve the values from controls, to intercept certain events and finally to provide filtering on the control. There are two predefined validators. wxGenericValidator and wxTextValidator. In addition to these you can create and use your own validators.

By default wxDev-C++ will use 'wxDefaultValidator' which is a NULL value meaning that no validation will be carried out on this control. To alter the validator used by wxDev-C++ you need to click on the button [...] which appears in the box next to the property 'Validator'



Figure 9.38 – The Validator edit button

This will bring up the Validator Editor

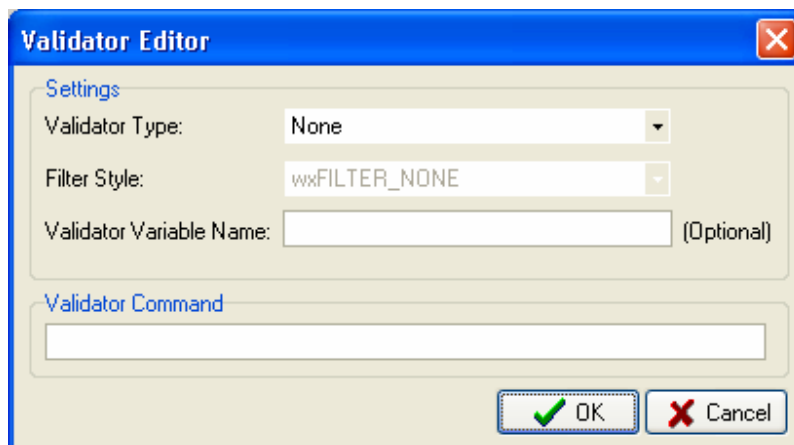


Figure 9.39 – The Validator Editor

This has two drop down boxes, Validator Type and Filter Style. The Validator Type allows the user to choose from wxTextValidator or wxGenericValidator.

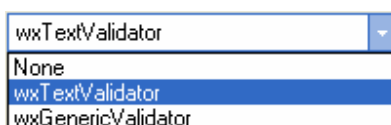


Figure 9.40 – The Validator Type selector

If you choose wxGenericValidator then the Filter Style will be disabled, since the Generic Validator only supports data transfer, not validation or filtering. For wxTextValidator it is possible to choose a Filter Style.

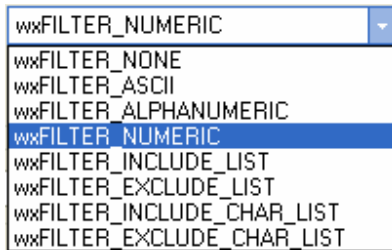


Figure 9.41 – The Filter Style selector

The filter styles are as follows

Filter Name	Purpose
wxFILTER_NONE	Provides no filtering
wxFILTER_ASCII	Only allows ASCII characters to be entered
wxFILTER_ALPHANUMERIC	Only allows letters and numbers to be entered
wxFILTER_NUMERIC	Only allows numbers to be entered
wxFILTER_INCLUDE_LIST	Allows the user to specify a list of acceptable strings
wxFILTER_EXCLUDE_LIST	Allows the user to specify a list of non acceptable strings
wxFILTER_INCLUDE_CHAR_LIST	Allows the user to specify a list of acceptable characters
wxFILTER_EXCLUDE_CHAR_LIST	Allows the user to specify a list of non acceptable characters

For the last four filter styles you will need to supply and attach the list of strings/characters yourself.

The final option is Validator Variable Name. This value is the name of a wxString. A pointer to this string will be used to supply the control with values and in turn will set the string to the controls value.

The last box Validator Command shows the code that is created by setting these options and is not something you would need to alter yourself.

The Components

Sizers

The components that raise the most questions are the sizers. They have no counterpart in traditional windows programming. However you will find similar components in Java and some other GUI libraries.

Sizers always have a parent control and they are designed to get larger and smaller when their parent alters size. At the same time they contain other controls or sizers and query these for their minimal size when they resize. Sizers themselves are non-visible controls.

But what is the benefit of this somewhat awkward method of laying out forms and dialogs? There are many benefits. The first is that on different platforms controls are different sizes. For example GTK uses a larger font than Windows by default. There for a button on Windows would be too small on GTK. Therefore part of the text might be lost, or the button might be bigger which will throw out your carefully arranged dialog. A second benefit is when the user resizes the frame. What happens now? If you have put all the controls exactly where you want them, they will stay there and if the frame is made smaller it could cover them. If it is enlarged there could be large blank areas on the frame. Sizers ensure that the controls react to the frame resizing in predetermined ways.

There are four types of sizer. These are the `wxBoxSizer`, `wxStaticBoxSizer`, `wxGridSizer` and `wxFlexGridSizer`. Each of these has a particular purpose and we will discuss these after we consider the properties that all of these have in common.

Common Sizer Properties

Alignment

This property affects how the control is displayed within the sizer. It can have any combination of the following 8 values (although only a few combinations make sense e.g. `wxALIGN_TOP` and `wxALIGN_LEFT`).

Flag name	Purpose
<code>wxALIGN_TOP</code>	The control will be aligned to the top edge of the sizer
<code>wxALIGN_LEFT</code>	The control will be aligned to the left edge of the sizer
<code>wxALIGN_RIGHT</code>	The control will be aligned to the right edge of the sizer
<code>wxALIGN_BOTTOM</code>	The control will be aligned to the bottom edge of the sizer
<code>wxCENTER</code>	The control will be centred horizontally and vertically within the sizer
<code>wxCENTER_HORIZONTAL</code>	The control will be centred horizontally within the sizer

wxCENTER_VERTICAL	The control will be centred vertically within the sizer
wxEXPAND	The control will expand to take up all the available space.

To see how these flags work we will create a new project.

- Create a new frame project
- Call it SizerAlignment
- Save it in a new folder called SizerAlignment
- Add a resizable border and maximize button to the default settings

You should now have a default frame.

- Click on the Box Sizer component then click on the Frame
- Change the sizer Alignment property to wxEXPAND
- Change the sizer Orientation property to wxVertical
- Change the sizer Stretch Factor property to 1

We now need to add another box sizer to this one.

- Select the box sizer you have already added
- Now select another box sizer from the component palette
- Click on the existing box sizer to place it
- Alter the properties to be the same as the existing sizer

Finally we want a third box sizer

- Add the third box sizer the same way as the last one
- Set the properties to the same values except the Orientation set this to wxHorizontal

Now we need to add some controls for the sizer to work with

- Add 4 button controls to the second sizer
- Add 4 button controls to the third sizer
- Resize the frame so you can see all the buttons
- Set the Alignment property for WxButton1 to wxALIGN_LEFT
- Set the Alignment property for WxButton2 to wxALIGN_RIGHT
- Set the Alignment property for WxButton3 to wxALIGN_CENTER_HORIZONTAL
- Set the Alignment property for WxButton4 to wxEXPAND
- Set the Alignment property for WxButton5 to wxALIGN_TOP
- Set the Alignment property for WxButton6 to wxALIGN_CENTER_VERTICAL
- Set the Alignment property for WxButton7 to wxALIGN_BOTTOM

Set the Alignment property for WxButton8 to wxCENTER

You should now have something that resembles this.

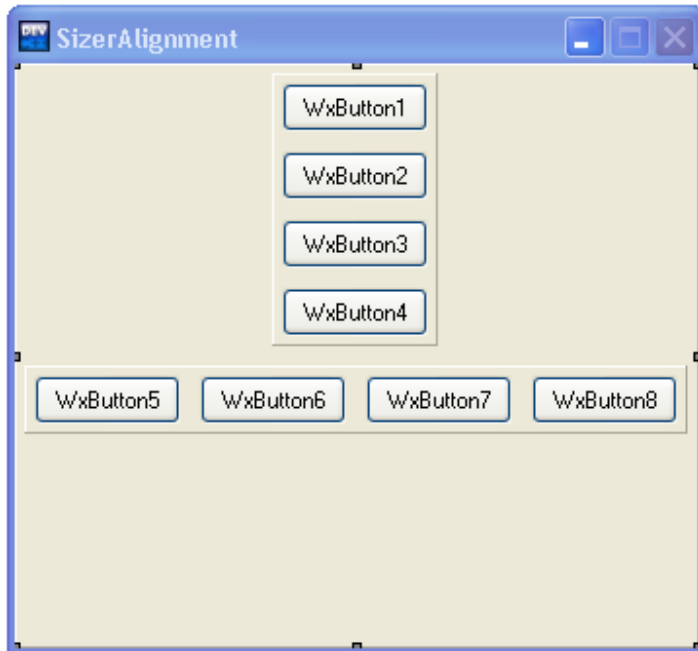


Figure 9.42 – The SizerAlignment project in design view

Press <F9> to compile and run the project

The result of running the project should be something approaching this

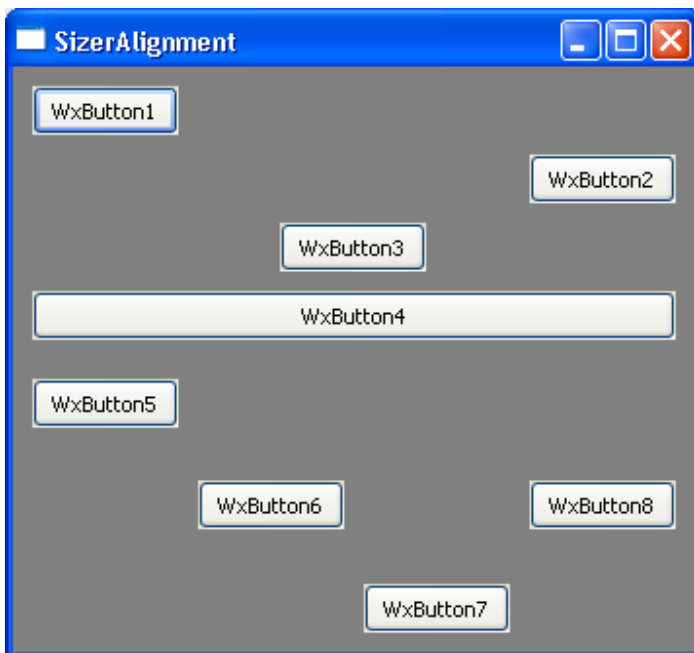


Figure 9.43 – The SizerAlignment project at runtime

The first thing you will realise is that the result looks nothing like the project at design time. This is one of the areas that wxDev-C++ is especially weak in. The sizers look nothing like the end result. The sizers are coloured in design view, but transparent at runtime. They don't resize the components at design time either.

But enough of picking on wxDev-C++. Play around with the sample. Try resizing the frame, you will notice that the buttons move according to the rules the flags set up. You will also notice that you cannot make the frame smaller than a certain size. As we go on hopefully you will realise the ease with which sizers can help layout your frames and dialogs.

Border

This is a sizer related property which affects how much padding the child of the sizer has between the walls of the sizer that contains it. This can be seen especially clearly around the expanded button in the previous example.

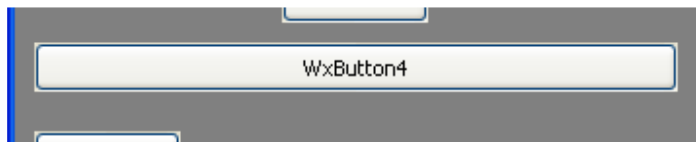


Figure 9.44 – Expanded button showing padding.

If you look under the properties for the button you will see that it has a border padding of 5. This is apparent by the spacing between it and the walls of the frame. You may think that the spacing looks like more than 5 pixels each side and you would be correct. For the sizers also have a border of 5 pixels.

Borders

This property is closely related to the Border property. The Border property specifies the thickness of the border. The Borders property is a set of flags which specify which side of the control receive this border. More than one flag can be set at a time, for example wxTOP and wxBOTTOM to pad the top and bottom but not the sides.

Flag name	Purpose
wxALL	All sides receive padding
wxBOTTOM	The bottom side of the control receives padding
wxLEFT	The left side of the control receives padding
wxRIGHT	The right side of the control receives padding
wxTOP	The top of the control receives padding

I leave it to you to play around with the previous sample to see the effect of border padding.

Stretch Factor

This property only affects controls in wxBoxSizers. You may have noticed that controls in the sample application were aligned in their parent sizers orientation. Thus Button4 aligned horizontally. The Stretch Factor alters how they should react to enlargement in the other dimension. The Stretch Factor is a proportional setting. If it is zero the controls will be unaffected in the other dimension. If it is above zero then the number is taken as a proportion. For example if you have two buttons in a sizer with a stretch factor of 1 then they will expand to take half the free space each.

To test this change the previous example so that buttons 3 and 5 have a stretch factor of 1 and button 4 has a stretch factor of 2. Now play with the compiled frame. You should see that Button5 stretches to fill the cell it is in. Button4 stretches twice as much as Button3.

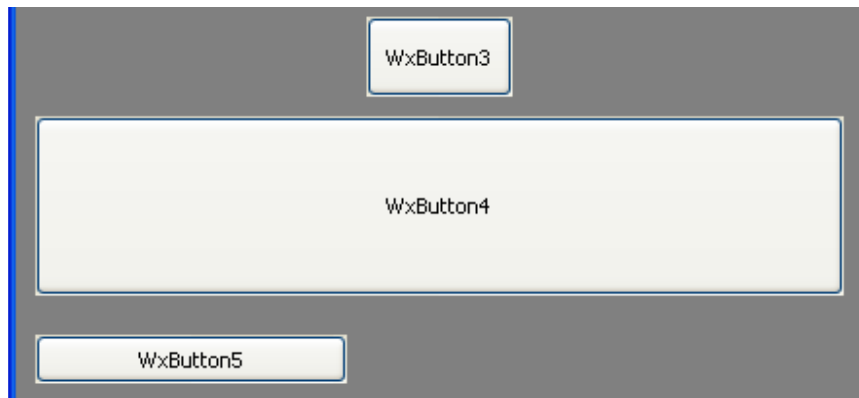


Figure 9.45 – The effect of using Stretch Factors

The Sizers

As previously mentioned there are 4 sizers and we will examine each one and see how they differ.

wxBoxSizer

We have pretty much covered the abilities of the wxBoxSizer in the last section. Box sizers are designed to layout their children in one dimension, either horizontally or vertically. For more complex layout they can be placed inside one another.

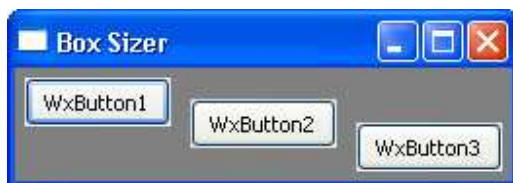


Figure 9.46 – A horizontal wxBoxSizer with differently aligned children

wxStaticBoxSizer

The wxStaticBoxSizer is the same as the box sizer with the exception that it draws a static box around the child controls. It has one additional property Caption. You will notice that the form designer doesn't handle the drawing of the static box sizer very well.

Caption

The caption property sets a caption in the top of the static box to explain its purpose.

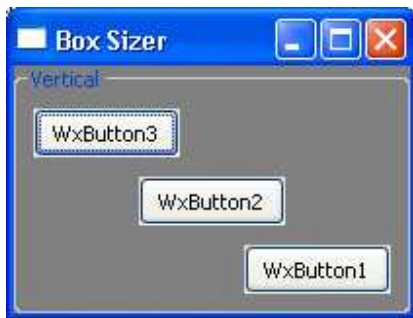


Figure 9.47 – A wxStaticBoxSizer with caption and differently aligned children

wxGridSizer

The grid sizers are two dimensional containers. The wxGridSizer gives all controls the same size container, the size of which is set by the minimal size of the largest control. It is suited for when you have many controls that you want laid out in a regular fashion. Since its purpose differs from box sizers it has extra properties.

Column Spacing

This sets the gap between the columns in the sizers

Columns

This sets the number of columns the grid contains

Height

Ignore this property, sizers do not have a height property

Left

Ignore this property, sizers do not have a leftt property

Row Spacing

This sets the gap between the rows in the sizers

Rows

This sets the number of rows the grid contains

Width

Ignore this property, sizers do not have a width property

Top

Ignore this property, sizers do not have a top property

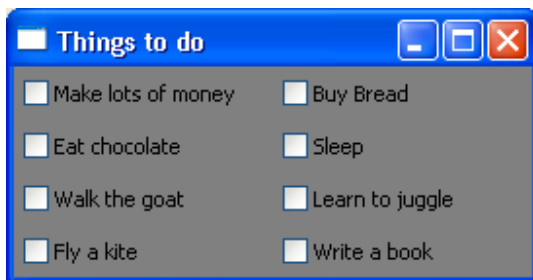


Figure 9.48 – The wxGridSizer with left aligned controls

wxFlexGridSizer

The wxFlexGridSizer is like the grid sizer. However the flex grid sizer does not specify that each cell must be the same size. Instead each cell on one row must be the same size and each cell in a column must be the same size. The flex grid sizer has the same properties as the grid sizer.

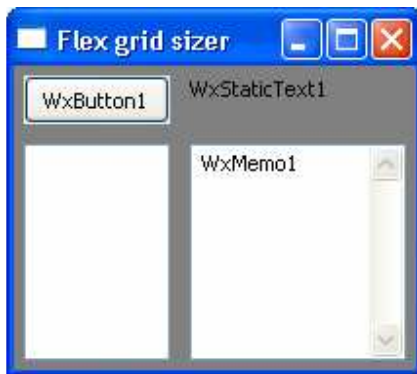


Figure 9.49 – The wxFlexGridSizer with different sized cells

This concludes our discussion on sizers. There is more on the topic which can be found in the wxWidgets help manual under sizers and the section sizers overview. There is also a handy guide available at <http://neume.sourceforge.net/sizedemo/>. We will be using sizers in our demonstration application at the end of this chapter.

Controls

The Controls palette in wxDev-C++ contains a ragbag assortment of common controls. For convenience I am going to break them down into 4 groups. Text controls, Buttons, Choices and the rest. I will start with the text controls.

Text Controls

I have grouped all the controls that are used for displaying text together in this section.

Static Text

A wxStaticText control is designed when you want to display one or more lines of text.

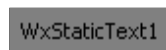


Figure 9.50 – A wxStaticText control

The Static Text control has these unique properties.

Label

This is the text that appears on the text control.

Label Style

This control uses the following flags to define its style

Flag Name	Purpose
wxALIGN_CENTER	This aligns the text to the center of the control
wxALIGN_LEFT	This aligns the text to the left of the control
wxALIGN_RIGHT	This aligns the text to the right of the control
wxST_NO_AUTORESIZE	This stops the control resizing its to fit its contents

Edit

The Edit control is a wxTextCtrl with the Multiline flag set to false. It allows users to display or edit a single line of text.



Figure 9.51 – A wxTextCtrl with multiline style set to false

The Edit control has these unique properties.

Edit Style

This control uses the following flags to define its style

Flag Name	Purpose
wxTE_AUTO_URL	Highlights URLs and sends URL events when the mouse is over them.
wxTE_BESTWRAP	Causes the text in the control to wrap at word or other character boundaries.
wxTE_CAPITALIZE	Causes the first letter to be capitalised.
wxTE_CENTRE	Text in this control will be centered.
wxTE_CHARWRAP	Wrap lines that are too long to be shown entirely.
wxTE_DONTWRAP	Don't wrap at all, show a horizontal scrollbar instead.
wxTE_LEFT	Text in this control will be left justified.
wxTE_MULTILINE	This flag allows the controls to show multiple lines of text.
wxTE_NOHIDSEL	This flag causes the selected text to be displayed even when the control loses focus.
wxTE_PASSWORD	If this flag is set all text will show up as asterisks '*'.
wxTE_PROCESS_ENTER	Setting this flag means that an Enter event will be triggered when the Enter key is pressed, otherwise the Enter key event will be used internally.
wxTE_PROCESS_TAB	Normally the TAB key is used to change the focus from the current control to another control. This flag alters that behaviour and sends a TAB event to this control.
wxTE_READONLY	This flag prevents users from editing the contents of this control.
wxTE_RICH	Uses the Rich Edit control under windows.
wxTE_RICH2	Uses the Rich Edit control version 2 or 3 under windows.
wxTE_RIGHT	Text in this control will be right justified.

MaxLength

Sets the maximum number of characters that can be entered into this control. Causes an event to be triggered if the user tries to enter more than this. It adds a line like the following in the `CreateGUIControls()` function.

```
WxMemo3->SetMaxLength( 20 );
```

Text

This allows you to set a line of text that will appear in the control, by default this is the controls name.

Memo

The Memo control is exactly the same control as the Edit control. The only difference is that the MultiLine property is set to true.

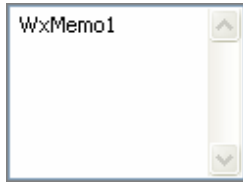


Figure 9.52 – A wxTextCtrl with multiline style set to true

The Edit control has these unique properties.

Edit Style

These are the same options as for the Edit Control, see Edit for the meanings.

Load From File

This property brings up a file selection dialog to choose the file to display in the memo. Adds a line like the following in the `CreateGUIControls()` function.

```
WxMemo3->LoadFile("C:/Dev-Cpp/copying.txt");
```

Care needs to be taken since this adds an absolute path, so the file needs to be in the same place on the users system.

Maximum Length

This has the same meaning as the Edit Controls MaxLength property, see Edit for the meaning.

Strings

This option brings up a dialog that allows you to enter the text that will appear in the control at runtime. However if you have already chosen to load from file any alterations you make here will be overridden.

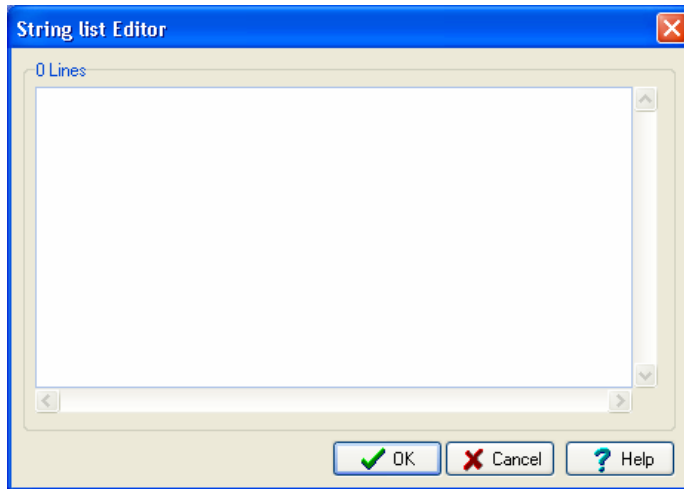


Figure 9.53 – The String Editor

The String Editor is used for a number of controls. The gotcha with the memo control is that if you enter several lines of text then compile the program you will see all the text jumbled together. The reason for this is that the Editor doesn't add the new lines into the resulting string. You need to put the '\n' for yourself.

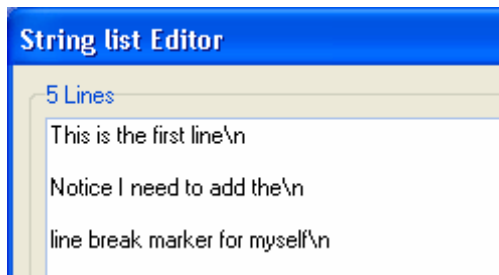


Figure 9.54 – The String Editor showing manual entry of line breaks

Rich Text Ctrl

The wxRichTextCtrl is a rich text control (meaning that it can show different styles of text and images at the same time). It is built using wxWidgets and not relying on the underlying native controls.



Figure 9.55 – The wxRichTextCtrl

The Rich Text control has these unique properties.

Load From File

This property is the same as the one for the wxTextCtrl. See Memo for more details.

Maximum Length

This property is the same as the one for wxTextCtrl. See Edit for more details.

RichText Style

This control uses the following flags to define its style

Flag Name	Purpose
wxRE_MULTILINE	Setting this flag allows you to use more than one line of text.
wxRE_READONLY	Setting this flag prevents the user from editing the contents.

Strings

This property has the same meaning as for the Memo control. See Memo for more details.

Styled Text Ctrl

The wxStyledTextCtrl is a wrapper around the Scintilla control. It provides many powerful features for writing text editors, such as line numbering, source code colourization, and much more. Support for this control is at present very limited in wxDev-C++.

```
1 <!-- Created us:  
2     http:\\skil  
3 <!-- Copyright  
4  
5     <THEME  
6         autho:  
7         autho:  
8         copyr:
```

Figure 9.56 – The wxStyledTextCtrl

The Styled Text control has these unique properties.

Load From File

This property has the same meaning as the Memo control. See Memo for more details.

Strings

This property has the same meaning as the Memo control. See Memo for more details.

Hyper Link Ctrl

The wxHyperlinkCtrl is similar to the static text control, except it displays a hyperlink. You are able to specify the URL it points to, the text it contains and how it reacts to the mouse.



Figure 9.57 – The wxHyperlinkCtrl

Hover Color

This option sets the colour of the text when the mouse hovers over it.

HyperLink Style

This control uses the following flags to define its style

Flag Name	Purpose
wxHL_CONTEXTMENU	Pops up a context menu that allows the user to copy the url (not the label) to the clipboard

Label

This option sets the text that will be shown on the control.

Normal Color

This sets the colour of the controls text before it has been clicked and while the mouse is not interacting with it.

Visited Color

This sets the colour of the controls text after it has been clicked upon.

Wx_URL

This property is a string that sets the URL that the control points to.

Buttons

Button

The wxButton is one of the basic elements of a GUI. It usually displays text describing its purpose.



Figure 9.58 – The Button control

Button Styles

This control uses the following flags to define its style

Flag Name	Purpose
wxBU_BOTTOM	Aligns the label to the bottom of the button
wxBU_EXACTFIT	Creates a button that is as small as possible
wxBU_LEFT	Aligns the label to the left of the button
wxBU_RIGHT	Aligns the label to the right of the button
wxBU_TOP	Aligns the label to the top of the button

Default

Sets this button to be the one with the focus which means it will be pressed when the Enter key is pressed.

Label

This property sets the text that will be displayed on the button.

Bitmap Button

A bitmap button is a button that displays a bitmap that usually shows its purpose.

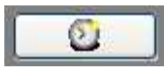


Figure 9.59 – The wxBitmapButton control

Bitmap

This option sets the bitmap that is displayed on the button. This bitmap is used for all the button states.

Button Styles

This control uses the following flags to define its style

Flag Name	Purpose
------------------	----------------

wxBU_AUTODRAW	If this flag is set the button will be drawn as flat style
wxBU_BOTTOM	Aligns the bitmap label to the bottom of the button
wxBU_LEFT	Aligns the bitmap label to the left of the button
wxBU_RIGHT	Aligns the bitmap label to the right of the button
wxBU_TOP	Aligns the bitmap label to the top of the button

Default

This property is the same as the Default value for the Button. See Button for more details.

Toggle Button

The wxToggleButton is a button that stays depressed once clicked.

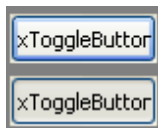


Figure 9.60 – The wxToggleButton in unpressed and pressed state

The Toggle Button has no extra properties.

Spin Button

The wxSpinButton has two arrow buttons used to increment or decrement an integer value.



Figure 9.61 – The wxSpinButton control

Max

This is an integer value that sets the maximum value this control can take.

Min

This is an integer value that sets the minimum value this control can take.

Orientation

This control uses the following flags to control its orientation

Flag Name	Purpose
wxSP_HORIZONTAL	Sets the controls orientation to horizontal, Arrows point left and right
wxSP_VERTICAL	Sets the controls orientation to vertical, Arrows point up and down

up and down

Spin Button Style

This control uses the following flags to define its style

Flag Name	Purpose
wxSP_ARROW_KEYS	User can use the arrow keys to alter the values
wxSP_WRAP	The controls value will wrap if it exceeds its maximum or minimum values

Choices

CheckBox

The wxCheckBox control is a control that can display an on or off state. Optionally it can have a third or undefined state.

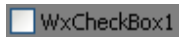


Figure 9.62 – The wxCheckBox control

Caption

This property is a string which defines the controls label

Checkbox Styles

This control uses the following flags to define its style

Flag Name	Purpose
wxALIGN_RIGHT	Puts the text label on the left and the check box on the right
wxCHK_2STATE	Creates a 2 state check box
wxCHK_3STATE	Creates a 3 state check box
wxCHK_ALLOW_3RD_STATE_FOR_USER	Allows the user to set the check box to the third state, otherwise it must be done within code

Checked

This property has a true or false value which defines if the control is created already checked.

Choice

The wxChoice control presents the user with a drop down box from which they can select a string.

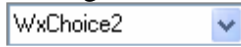


Figure 9.63 – The wxChoice control

Index

This is the index of the currently selected item. The numbering starts with the first item numbered 0.

Items

This property brings up the string editor to allow you to set the string items contained by this control. Unlike the memo control you don't need to add new line markers to the end of the strings. Each line in the string editor will become a separate item in the Choice Box.

Radio Button

A wxRadioButton is usually a round control that occurs in groups. Only one button of the group can be selected at a time. If a new button is selected the previously selected button is deselected.

The first radio button in a group is marked with the wxRB_GROUP flag. Thereafter every radio button belongs to the same group until another radio button is placed marked with the wxRB_GROUP flag.



Figure 9.64 – The wxRadioButton control

Caption

This property is the same as the Caption property in CheckBox. See CheckBox for more details.

Checked

This property is the same as the Checked property in CheckBox. See CheckBox for more details.

Radio Button Style

This control uses the following flags to define its style

Flag Name	Purpose
wxBUTTON_GROUP	Marks this button as part of a group
wxBUTTON_SINGLE	Marks this button as not being part of a group

Combo Box

The wxComboBox is a more powerful version of the wxChoice. It combines a text control with a list control and allows users to select one item out of a list. All items in the combo box are numbered starting from zero.

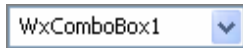


Figure 9.65 – The wxComboBox control

Combobox Style

This control uses the following flags to define its style

Flag Name	Purpose
wxCB_DROPDOWN	The list of choices are displayed in a drop down box
wxCB_READONLY	Only the choices in the drop down list can be selected
wxCB_SIMPLE	The list of choices is permanently displayed
wxCB_SORT	The choices are sorted into alphabetical order

Edit Style

The combo box control comes with a list of edit styles. The only one that has any effect is wxTE_PROCESS_ENTER and this means the same as the flag with the same name in the edit control. See Edit for more details.

Items

This property has the same meaning as the one in the choice control. See Choice for more details.

Text

This property is a string value that sets the text displayed in the combo box. If the flag wxCB_READONLY is set to true, then this string must appear in the Items list otherwise it will be ignored.

List Box

A wxListBox is used in a similar way to a wxChoice or wxCombo, in that it allows a user to choose from a list of items. The main difference is that the contents are always displayed. It is also possible to design it so that the user can select multiple items.

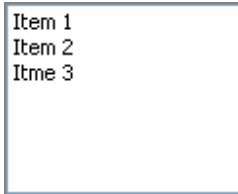


Figure 9.66 – The wxListBox control

Listbox Style

This control uses the following flags to define its style

Flag Name	Purpose
wxLB_ALWAYS_SB	Always show a vertical scroll bar
wxLB_HSCROLL	Create horizontal scroll bar when the contents are too wide
wxLB_NEEDED_SB	Only show a vertical scroll bar when all the contents can not be seen
wxLB_SORT	The items are sorted into alphabetical order

Selection Mode

This control uses the following flags to define its selection style

Flag Name	Purpose
wxLB_SINGLE	Creates a single selection list
wxLB_MULTIPLE	Creates a multiple selection list, where the user can toggle contents between selection states
wxLB_EXTENDED	Creates the extended multiple selection list, user can select multiple items while holding down the Shift key

Strings

This has the same meaning as for the Choice control. See Choice for more details.

List Ctrl

The wxListCtrl is a powerful control which allows you to display lists in several different manners. This can be a multi column list, a report view, an icon view or a small icon view. List Controls are used in applications such as Windows Explorer which allow the user to view the files within a folder in different ways.

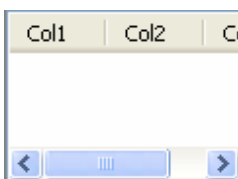


Figure 9.67 – The wxListCtrl control

Columns

The Columns property opens a dialog to enable you to add columns to the ListCtrl.

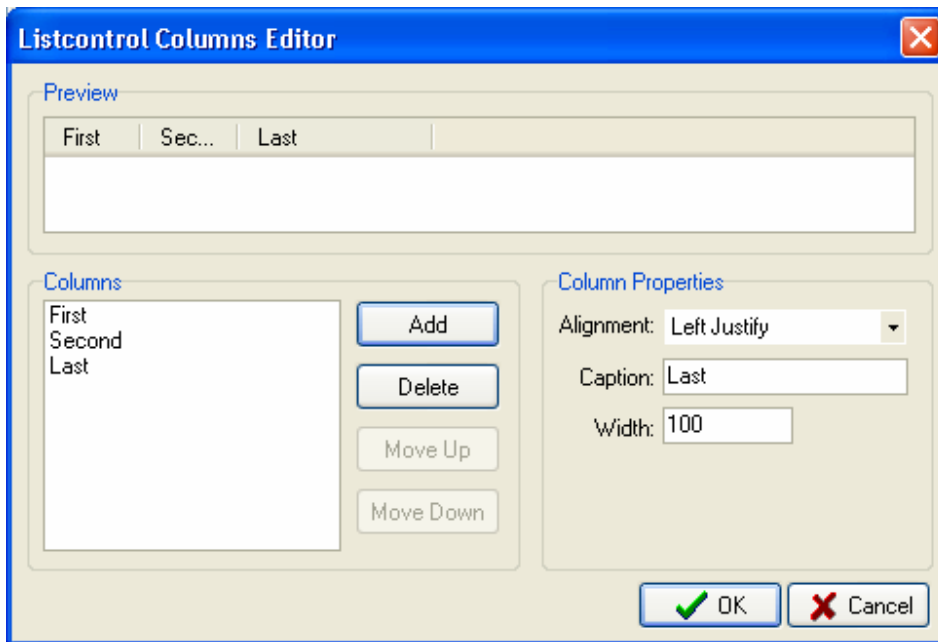


Figure 9.68 – The ListControl Column Editor

To use this editor you need to use the Column Properties section.

Alignment

This gives you the options to set the justification for the column.

Alignment	Purpose
Left Justify	Sets the text alignment to the left
Center	Sets the text alignment to the center
Right Justify	Sets the text alignment to the right

Caption

This string property sets the caption for new column.

Width

This integer property sets the width of the column

Adding a New Column

Once you have set the properties for the new column, you click the [Add] button to add the column to the control. This will be shown in the preview.

Deleting an Existing Column

To delete a column you have previously created select the column by its caption in the box under the caption Columns.

Rearranging an Existing Column

To re-order previously created columns select the column you wish to move by clicking on the corresponding caption in the box under the caption Columns.

At present there is no way to alter the properties of an existing column, instead you need to delete and recreate it.

Items

At present the Items property is unused.

Listview Style

This control uses the following flags to define its style

Flag Name	Purpose
wxLC_ALIGN_LEFT	Icons align to the left
wxLC_ALIGN_TOP	Icons align to the top
wxLC_AUTOARRANGE	Icons arrange themselves
wxLC_EDIT_LABELS	Labels are user editable
wxLC_HRULES	Draws light horizontal lines between rows when wxLC_REPORT is set
wxLC_NO_HEADER	If the flag wxLC_REPORT is set, then no column headers will be shown
wxLC_NO_SORT_HEADER	Prevents the header acting as a button
wxLC_SINGLE_SEL	Only single items can be selected, default is multiple selection
wxLC_SORT_ASCENDING	Sort in ascending order (needs a comparison callback setting using the SortItems() method
wxLC_SORT_DESCENDING	Sort in descending order (needs a comparison callback setting using the SortItems() method
wxLC_VRULES	Draws light vertical lines between rows when wxLC_REPORT is set

Listview View

This control uses the following flags to define its view style

Flag Name	Purpose
wxLC_ICON	Items are viewed as large icons with optional labels
wxLC_SMALL_ICON	Items are viewed as small icons with optional labels
wxLC_LIST	Multi-column view with optional icons
wxLC_REPORT	Single or multi-column view with optional headers for the columns
wxLC_VIRTUAL	Can only be used with wxLC_REPORT the application provides the text on demand, used for huge amounts of data

Radio Box

The wxRadioBox is designed to display a group of mutually exclusive options surrounded with a static border and an optional caption.

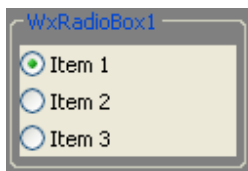


Figure 9.69 – The wxRadioBox control

Caption

This string property defines the label that appears in the static border.

Items

This property has the same meaning as for the Choice control. See Choice for more details.

Major Dimension

Depending on the flag setting in Radiobox Style this setting specifies the maximum number of rows or columns in a 2 dimensional array. See Radiobox Style for more details.

Radiobox Style

This control uses the following mutually exclusive flags to define its style

Flag Name	Purpose
wxRA_SPECIFY_COLS	If this setting is chosen then the Major Dimension property will specify the maximum number of columns

wxRA_SPECIFY_ROWS If this setting is chosen then the Major Dimension property will specify the maximum number of rows

Selected Button

This property calls the wxRadioButton function SetSelection to set the currently selected button.

Date Picker Ctrl

The wxDatePickerCtrl is a simple control that allows the user to set a date. This can be achieved by direct editing or via a popup window depending on which flags are set at creation time.

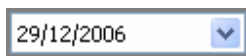


Figure 9.70 – The wxDatePickerCtrl control

Date

This allows you to set the date and time that will be displayed by default.

Picker Style

This control uses the following flags to define its style

Flag Name	Purpose
wxDP_ALLOWNONE	Allows the user to enter any date, even an invalid one
wxDP_DEFAULT	Creates an edit style to suit the platform (Spin buttons under Windows, drop down on other platforms)
wxDP_DROPDOWN	Creates a month calendar drop down
wxDP_SHOWCENTURY	Forces the century to be shown, otherwise the date is displayed in a format that suits the system
wxDP_SPIN	Don't show drop down calendar, edit using a spin button instead

Check List Box

The wxCheckListBox works just like a standard list box with the difference that the contents can be checked and unchecked.

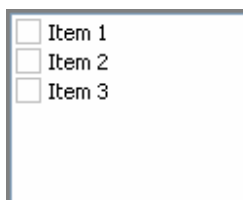


Figure 9.71 – The wxCheckListBox control

Items

This has the same meaning as for the Choice control. See Choice for more details.

Listbox Style

This control uses the following flags to define its style

Flag Name	Purpose
wxBL_ALWAYS_SB	Always show a vertical scroll bar
wxBL_HSCROLL	Create horizontal scroll bar when the contents are too wide
wxBL_NEEDED_SB	Only show a vertical scroll bar when all the contents can not be seen
wxBL_SORT	The items are sorted into alphabetical order

Selection Mode

This control uses the following flags to define its selection style

Flag Name	Purpose
wxBL_SINGLE	Creates a single selection list
wxBL_MULTIPLE	Creates a multiple selection list, where the user can toggle contents between selection states
wxBL_EXTENDED	Creates the extended multiple selection list, user can select multiple items while holding down the Shift key

Spin Ctrl

A wxSpinCtrl is a combination of a text control and a spin button control. It allows a user to alter the value by using the up or down buttons.



Figure 9.72 – The wxSpin control

Edit Style

This control uses the following flags to define the text control style

Flag Name	Purpose
wxTE_AUTO_URL	Highlights URLs and sends URL events when the mouse is over them.
wxTE_BESTWRAP	Causes the text in the control to wrap at word or other character boundaries.

wxTE_CAPITALIZE	Causes the first letter to be capitalised.
wxTE_CENTRE	Text in this control will be centered.
wxTE_CHARWRAP	Wrap lines that are too long to be shown entirely.
wxTE_DONTWRAP	Don't wrap at all, show a horizontal scrollbar instead.
wxTE_LEFT	Text in this control will be left justified.
wxTE_MULTILINE	This flag allows the controls to show multiple lines of text.
wxTE_NOHIDSEL	This flag causes the selected text to be displayed even when the control loses focus.
wxTE_PASSWORD	If this flag is set all text will show up as asterisks '*'.
wxTE_PROCESS_ENTER	Setting this flag means that an Enter event will be triggered when the Enter key is pressed, otherwise the Enter key event will be used internally.
wxTE_PROCESS_TAB	Normally the TAB key is used to change the focus from the current control to another control. This flag alters that behaviour and sends a TAB event to this control.
wxTE_READONLY	This flag prevents users from editing the contents of this control.
wxTE_RICH	Uses the Rich Edit control under windows.
wxTE_RICH2	Uses the Rich Edit control version 2 or 3 under windows.
wxTE_RIGHT	Text in this control will be right justified.

Maximum Value

This integer property controls the maximum value the user can enter. If they enter a higher value the control will only return the maximum allowed.

Minimum Value

This integer property controls the minimum value a user can enter.

Spin Control Style

This control uses the following flags to define the spin button style

Flag Name	Purpose
wxSP_ARROW_KEYS	User can use the arrow keys to alter the values
wxSP_WRAP	The controls value will wrap if it exceeds its maximum or minimum values

Owner Drawn Combo Box

The wxOwnerDrawnComboBox is a combo box that allows you to control the way it looks. For example you could use it to draw an icon before the text. However to use it in this manner you would need to create a new control that derives from it. This control is therefore of little use, unless you have a derived control in which case you can change the base class to use your own variant.

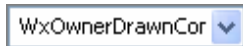


Figure 9.73 – The wxOwnerDrawnComboBox control

Calendar Ctrl

The wxCalendarCtrl allows the user to select a day, month and year.

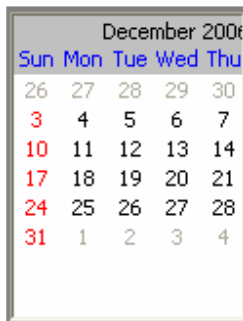


Figure 9.74 – The wxCalendarCtrl control

Calendar Style

This control uses the following flags to define the text control style

Flag Name	Purpose
wxCAL_MONDAY_FIRST	Sets Monday to be the first day of the week
wxCAL_NO_MONTH_CHANGE	If this option is set the user will be unable to change the month
wxCAL_NO_YEAR_CHANGE	If this option is set the user will be unable to change the year
wxCAL_SEQUENTIAL_MONTH_SELECTION	If this option is set the month year selection is shown in a compact style
wxCAL_SHOW_HOLIDAYS	If this option is set the holidays will be highlighted in the control
wxCAL_SHOW_SURROUNDING_WEEKS	If this option is set the months previous and following weeks will be

<code>wxCAL_SUNDAY_FIRST</code>	shown Sets Sunday to be the first day of the week
---------------------------------	--

Selected Date

This property sets the date the calendar control is set to by default

The Rest

Tree Ctrl

The `wxTreeCtrl` is useful for displaying information in a hierarchical manner. The most common uses for this are the drive and directory side bar in Windows Explorer or the history bar in a browser. The control allows you to customise the appearance of the twist buttons and whether icons are displayed. Currently `wxDev-C++` has no inbuilt method to help with displaying icons.

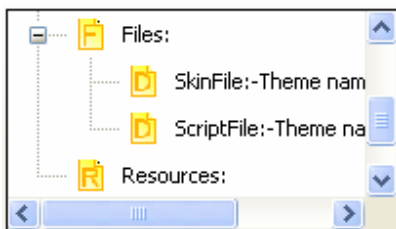


Figure 9.75 – The `wxTreeCtrl` control

This control has the following unique properties.

Items

I guess this property should produce an editor to allow you to add a list of items that will be displayed when the control is created. At present this property does nothing.

Tree Control Styles

This control uses the following flags to define the tree control style

Flag Name	Purpose
<code>wxTR_COLUMN_LINES</code>	This flag appears to have been removed in recent versions of <code>wxWidgets</code> . The IDE will ignore it if you try to use it
<code>wxTR_DEFAULT_STYLE</code>	Choose the best style for the current platform

wxTR_EDIT_LABELS	This flag allows the user to edit the tree control labels at runtime
wxTR_EXTENDED	Allow disjoint items to be selected
wxTR_FULL_ROW_HIGHLIGHT	When item is selected this flag ensures the whole row is highlighted not just the label
wxTR_HAS_BUTTONS	This flag draws + and – buttons by each item that contains children
wxTR_HAS_VARIABLE_ROW_HEIGHT	This flag causes the control to adjust to different height item
wxTR_HIDE_ROOT	This flag causes the root node to be hidden
wxTR_LINES_AT_ROOT	Show lines between root items as long as wxTR_HIDE_ROOT is true and wxTR_HIDE_LINES is false
wxTR_MULTIPLE	This flag allows the user to select multiple items
wxTR_NO_BUTTONS	Stops buttons from being drawn
wxTR_NO_LINES	Stop the vertical lines from being drawn
wxTR_ROW_LINES	This flag causes a contrasting border to be displayed between rows
wxTR_SHOW_ROOT_LABEL_ONLY	This flag appears to have been removed in recent versions of wxWidgets. The IDE will ignore it if you try to use it
wxTR_SINGLE	The default this allows only one item to be selected at a time
wxTR_TWIST_BUTTONS	Display twist buttons

Gauge

The wxGauge has two modes of operation. The first displays a gauge which displays the progress of time or an operation. The second is used to display activity for example during a long operation that you have no means of measuring.



Figure 9.76 – The wxGauge control

Gauge Styles

This control uses the following flags to define the gauge style

Flag Name	Purpose
------------------	----------------

<code>wxGA_MARQUEE</code>	This flag was created for use with a patch written by Joel Low. However this flag does not appear in the standard distribution and should not be used
<code>wxGA_SMOOTH</code>	Creates a gauge that updates by one pixel width at a time

Orientation

This property has one of two values `wxGA_VERTICAL` or `wxGA_HORIZONTAL` this affects the orientation of the gauge.

Range

This integer property sets the maximum value this gauge can display.

Value

This integer property sets the current value displayed by the gauge.

Scroll Bar

The `wxScrollbar` control provides a scrollbar which sends events when the user moves it. It is up to you to catch the events and interpret them as you wish.



Figure 9.77 – The `wxScrollbar` control

Orientation

This property has one of two values `wxSB_VERTICAL` or `wxSB_HORIZONTAL` this affects the orientation of the scrollbar.

Static Box

The `wxStaticBox` control is designed to logically group components together. It draws a border around the controls it contains and contains a caption.



Figure 9.78 – The `wxStaticBox1` control

Caption

This string property sets the name that will be displayed within the border of the control.

Slider

The wxSlider is similar to the Scroll Bar control. It has a handle for the user to move to set the value. It can also be used to display a value visually.

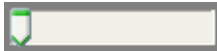


Figure 9.79 – The wxSlider control

Max

This integer property sets the maximum value this control can represent.

Min

This integer property sets the minimum value this control can represent.

Orientation

This property has one of two values wxSL_VERTICAL or wxSL_HORIZONTAL this affects the orientation of the scrollbar.

Selection Style

This control uses the following flags to define the selection style

Flag Name	Purpose
wxSL_SELRange	This flag allows the user to select a range on the gauge
wxSL_INVERSE	This flag inverts the minimum and maximum endpoints on the gauge

Tree Control Styles

This control uses the following flags to define the tree control style

Flag Name	Purpose
wxSL_AUTOTICKS	This flag causes the tick marks to be shown
wxSL_BOTTOM	Displays the ticks on the bottom
wxSL_LABELS	Displays minimum, maximum and value labels
wxSL_LEFT	Displays the ticks on the left and puts the slider into the upright position
wxSL_RIGHT	Displays the ticks on the right and

<code>wxSL_TOP</code>	puts the slider into the upright position Displays the ticks on the top
-----------------------	--

Static Line

The `wxStaticLine` control exists for the purpose of separating and grouping controls much in the same way as the `wxStaticBox` does.



Figure 9.80 – The `wxStaticLine` control

Orientation

This property has one of two values `wxLI_VERTICAL` or `wxLI_HORIZONTAL` this affects the orientation of the scrollbar.

Static Bitmap

The `wxStaticBitmap` is used to display small pictures. It has limitations that mean it cannot be used for large images on certain platforms.



Figure 9.81 – The `wxStaticBitmap` control

The Static Bitmap control has the following unique property.

Picture

This property allows you to select the image that will be displayed in the control. Clicking on the [...] button displays the following dialog.

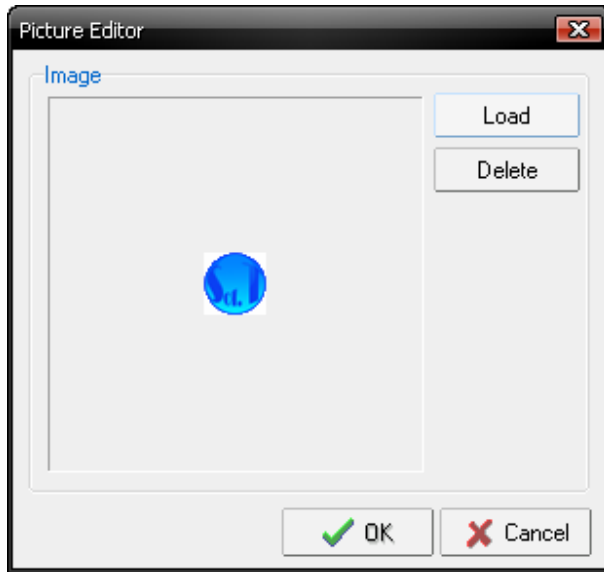


Figure 9.82 – The picture chooser dialog

Pressing the [Load] button produces an open file dialog to allow you to select an image file. Once you click the [OK] button the image is converted into an XPM image and saved in the projects Image folder.

Status Bar

The wxStatusBar control is a narrow bar which sits at the bottom of a frame. It can be divided into a number of separate fields which are used to display small amounts of information.

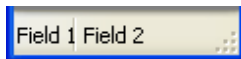


Figure 9.83 – The wxStatusBar control

The Status Bar control has the following unique properties.

Fields

Clicking on the [...] button in this property produces the field editor.

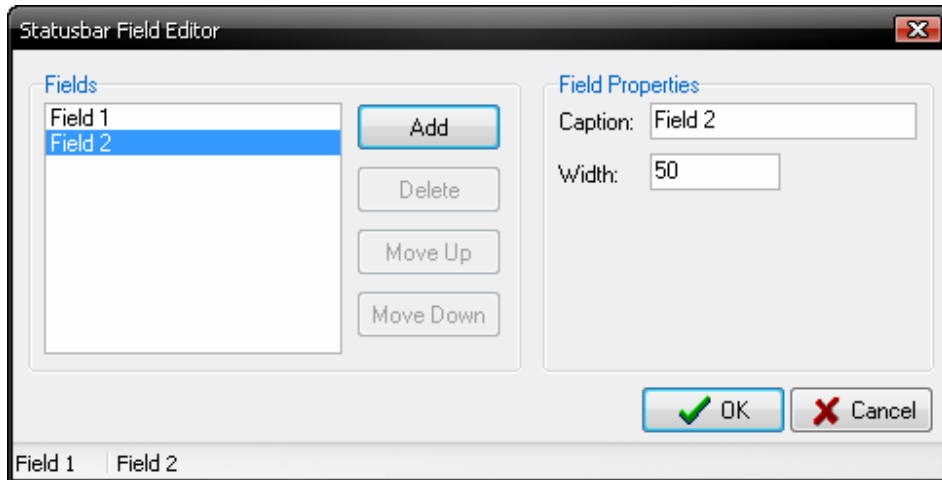


Figure 9.84 – The statusbar field editor

On the righthand side you can enter a caption to be displayed in this field as well as the size of the field. Clicking the [Add] button causes the field to be added to the list of fields on the left. The bottom of the control also displays a status bar with your fields add to it so you can see if the sizes are as you want them to be.

There is no provision to edit fields once you have created them, but you can rearrange them using the [Move Up] and [Move Down] buttons. You can also delete them with the [Delete] button.

Statusbar Styles

This control uses the following flag to define its style

Flag Name	Purpose
wxST_SIZEGRIP	This flag displays a sizing grip on the righthand side of the control (Windows 95 only)

Window

This category contains the controls that are used as parents to other controls.

Panel

The wxPanel control derives from wxWindow without adding much functionality. It is generally used as a container for other controls. It is also excellent for using as a base class to derive your own visual controls from.

One point to note is that if you add this to a frame in wxDev-C++ it will expand to fill the whole of the frame. This is the correct behaviour if the panel is the first or only component on the frame. However if you already have other controls on the frame it expand and cover them which is not the correct behaviour. To avoid this it is advisable to either add a panel or a sizer as your first component on the frame.



Figure 9.85 – The wxPanel

The Panel control has no extra properties.

Note Book

The wxNotebook is a control to hold multiple windows displaying one at a time. The displayed window is selected by clicking on its tab.

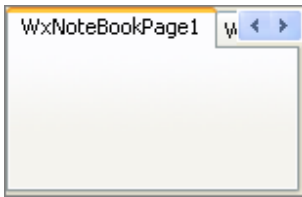


Figure 9.86 – The wxNotebook control

The notebook control has the following unique property.

Notebook Styles

This control uses the following flags to define the tree control style

Flag Name	Purpose
wxNB_BOTTOM	This flag causes the tabs to be displayed on the bottom
wxNB_FIXEDWIDTH	Makes all the tabs the same width (Windows only)
wxNB_LEFT	This flag causes the tabs to be displayed on the left
wxNB_MULTILINE	Tabs are displayed on multiple lines (Windows only)
wxNB_NOPAGETHEME	Draws tabs in a solid colour not as a gradient (Windows only)
wxNB_RIGHT	This flag causes the tabs to be displayed on the right

Note Book Page

A wxNotebook can take any wxWindows as a child page. wxDev-C++ simplifies the creation of notebooks by using wxPanel as the default control to use as a notebook page. If you wish to use other controls as notebook pages you will need to alter the Base Class property to match.

To edit the page for example to add new controls you need to click on the tab for the page you wish to edit. Then add controls to that page.

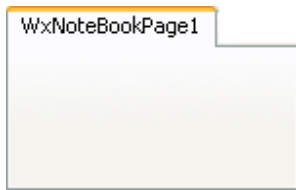


Figure 9.87 – The note book page

The control has the following unique property.

Label

This string property sets the caption that will appear on the tab for this page.

Grid

The wxGrid class is designed to work with tabular data. The most common use for this style of control is in spreadsheet programs such as Microsoft Excel.

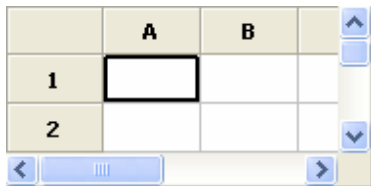


Figure 9.88 – The wxGrid control

The Grid control has the following unique properties.

Column Count

This integer property sets the number of columns contained in the grid.

Column Width

This integer property sets the default width of the columns in the grid.

Grid Selection

This property uses one of three exclusive flags to control

Flag Name	Purpose
wxGridSelectCells	Clicking on a cell causes that cell to be highlighted
wxGridSelectColumns	Clicking on a cell causes all cells in that column to be highlighted
wxGridSelectRows	Clicking on a cell causes all cells in the row to be highlighted

Label Column Width

This integer property sets the width of the column labels.

Label Row Height

This integer property sets the height of the row labels.

Row Count

This integer property sets the number of rows contained in the grid.

Row Height

This integer property sets the default height of the rows in the grid.

Scrolled Window

The wxScrolledWindow control is a window that has a client area which can be larger than the actual size of the window. In this case it will provide scrollbars which give you the control to move the contents of the client area.

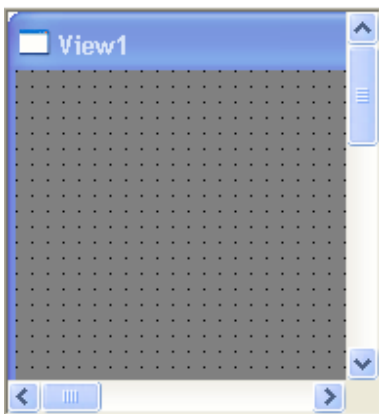


Figure 9.89 – A wxScrolledWindow with scrollbars to control its larger client area

The Scrolled Window control has the following unique property

Scrolled Window Styles

This property has just one flag

Flag Name	Purpose
wxRetained	This only has meaning on the Motif platform to speed up refreshing

Html Window

The wxHtmlWindow has the single purpose of allowing you to display simple HTML documents. It is able to render most HTML documents, though it seems to have problems with tables. It is not suitable for use with embedded scripting languages, CSS or other advanced web documents.

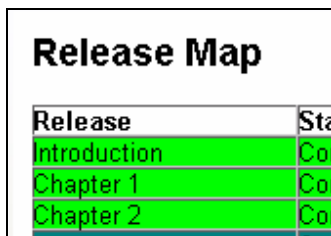


Figure 9.90 – The wxHtmlWindow displaying part of the project site

The Html Window control has the following unique property

HTML Window Styles

This property has the following flags.

Flag Name	Purpose
wxHW_NO_SELECTION	Prevent the user from selecting text
wxHW_SCROLLBAR_AUTO	Display scrollbars if the document is larger than the window
wxHW_SCROLLBAR_NEVER	Never display scrollbars

Splitter Window

The wxSplitterWindow controls two subwindows. It allows these to be resized via a sash handle between to two.

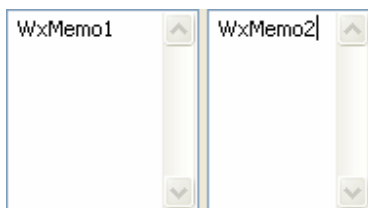


Figure 9.91 – The Splitter Window control

The Splitter Window control has the following unique properties.

Orientation

This property has one of two values `wxVERTICAL` or `wxHORIZONTAL` this affects the orientation of the scrollbar.

Sash Position

This integer property gives the initial position of the sash. If this value is positive, it specifies the size of the upper or left pane depending on orientation. If it is negative, its absolute value gives the size of the lower or right pane. A value of 0 will make both panes the same size. This will be set automatically as you add controls to the splitter window, however you can alter it to suit.

Splitter Window Styles

This property has the following flags.

Flag Name	Purpose
<code>wxSP_3D</code>	Draws 3D border and sash
<code>wxSP_3DBORDER</code>	Draws 3D border
<code>wxSP_3DSASH</code>	Draws 3D sash
<code>wxSP_BORDER</code>	Draws standard border
<code>wxSP_LIVE_UPDATE</code>	Resizes child windows when sash is moved, otherwise just a line is drawn until the sash is released
<code>wxSP_NO_XP_THEME</code>	On Windows XP control assumes pre-XP look
<code>wxSP_NO_BORDER</code>	Draws no border (the default setting)
<code>wxSP_PERMIT_UNSPPLIT</code>	Always allow the child panes to be unsplit

Toolbar

The Toolbar palette contains all the components that can be used with toolbars. The toolbar is a recognizable part of most GUI applications. For example here is a part of the toolbar in `wxDev-C++`.



Figure 9.92 – Part of the toolbar in `wxDev-C++`

To see how the toolbar designer works in `wxDev-C++` we will create a sample project and add a toolbar to it with one each of the components.

Open wxDev-C++
Select File|New|Project...
Select wxWidgets Frame
Change Project Name to “ToolbarSample”
Make sure C++ Project is selected
Click the [OK] button
Create a new folder in your projects
Call the folder “ToolbarSample”
Save the project in this folder
On the next dialog check the Resize Border and Max Button
Click the [Create] button

You will now have the usual blank frame.

Change the Component Palette to the Toolbar Palette. You will find that most of the components are already familiar to you from the Controls palette. We will start by looking at the Toolbar itself.

Tool Bar

The wxToolBar control can only be used on a wxFrame control. In the designer the toolbar will jump to the top of the form designer.



Figure 9.93 – The Toolbar control placed and selected on the form designer

To add controls to the toolbar you need to make sure the toolbar is selected as above, then select the tool you wish to add from the component palette. Now drop the control onto the toolbar. If you don't do this you will get the following warning.

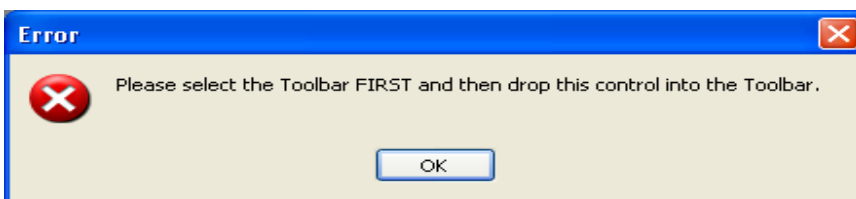


Figure 9.94 – Result of trying to drop a tool component somewhere other than the toolbar

Toolbar Styles

This control uses the following flags to define its style

Flag Name	Purpose
wxBTB_DOCKABLE	Creates a toolbar that can be docked and undocked (Only on Linux under GTK)

wxBT_FLAT	Specifies that the toolbar should have a flat appearance
wxBT_HORIZONTAL	Specifies a horizontal layout of controls
wxBT_HORIZ_LAYOUT	Specifies that text and images should be displayed alongside each other not stacked
wxBT_HORIZ_TEXT	This is a shortcut that is the same as specifying wxBT_HORIZ_LAYOUT and wxBT_TEXT
wxBT_NOALIGN	Windows only flag that affects the alignment with the frame.
wxBT_NODIVIDER	Specifies if a divider should be shown at the top of the toolbar
wxBT_NOICONS	Specifies that the images are not shown in toolbar buttons
wxBT_TEXT	Specifies that text should be displayed in the toolbar buttons
wxBT_VERTICAL	Specifies a vertical layout of controls

Let us add a toolbar to the frame.

Select the toolbar component and drop it onto the frame.
Alter the toolbar style wxBT_TEXT to true.

That is it for this control, so let us look at the others.

Tool Button

The tool button is similar to a wxBitmapButton, however it can only be placed in a toolbar. Depending on the toolbar settings it can display either a bitmap, text or both together.

Bitmap

This property is the same for the BitmapButton. See BitmapButton under Controls for more details.

Label

This property is a string that sets the label that will appear on the button

Type

This control uses the following flags to define its view style

Flag Name	Purpose
wxITEM_NORMAL	Creates a button which acts like a normal button

wxITEM_RADIO	Creates a button which forms a radio group, if one button of this type is selected the others are unselected.
wxITEM_CHECK	Creates a toggle style button, pressing it toggles whether it is selected or not.

Now let us try some of these buttons out in our sample project.

Drop 4 ToolButtons onto the Toolbar, remember to select the toolbar inbetween. Set the following properties.

WxToolButton1 - Design time properties

Bitmap	Browse to the Icon directory in the Dev-Cpp folder and select Smile.ico.
Label	Alter this to read 'Happy?'
Type	Make sure this is set to wxITEM_NORMAL

WxToolButton2 - Design time properties

Bitmap	Browse to the Icon directory in the Dev-Cpp folder and select Goofy.ico.
Label	Leave this empty
Type	Make sure this is set to wxITEM_RADIO

WxToolButton3 - Design time properties

Bitmap	Browse to the Icon directory in the Dev-Cpp folder and select Ufo.ico.
Label	Leave this empty
Type	Make sure this is set to wxITEM_RADIO

WxToolButton4 - Design time properties

Bitmap	Browse to the Icon directory in the Dev-Cpp folder and select Food.ico.
Label	Alter this to read 'Hungry?'
Type	Make sure this is set to wxITEM_CHECK

You should end up with something resembling this.



Figure 9.95 – The toolbar with buttons at design time

You will notice that it looks strange, the buttons are not quite aligned with the toolbar and the images are cut-off, also the labels don't appear. Don't worry it won't look like this at runtime. In fact let us prove that.

Press <F9> to compile and run.

You should now see a toolbar that looks like this.

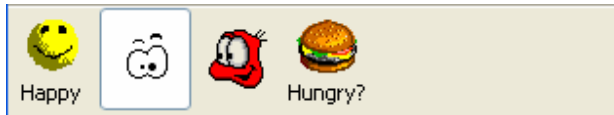


Figure 9.96 – The toolbar sample with various buttons

You will immediately notice that the buttons and the toolbar have resized to accommodate the size of the icons and the text. Play around with it and see what the different buttons do.

Separator

The separator has no properties and merely creates a gap between controls. Depending on the platform this gap may appear only as a space or it may be drawn. The separator control at present doesn't work well with the toolbar style `wxTB_TEXT`. It is not recommended to use them together.

Edit

The Edit control is just a `wxTextCtrl`. See Edit under Controls for more details.

CheckBox

The CheckBox control is just a `wxCheckBox`. See CheckBox under Controls for more details.

RadioButton

The RadioButton is just a `wxRadioButton`. See RadioButton under Controls for more details.

ComboBox

The ComboBox control is just a `wxComboBox`. See ComboBox under Controls for more details.

SpinCtrl

The SpinCtrl control is just a `wxSpinCtrl`. See SpinCtrl under Controls for more details.

Drop one each of these controls, other than the separator onto the toolbar sample in order. You may need to stretch out the frame to see all the controls. You will end up with something like this.



Figure 9.97 – The finished toolbar sample in the design view

We will now compile it once more.

Press <F9> to compile and run the sample.

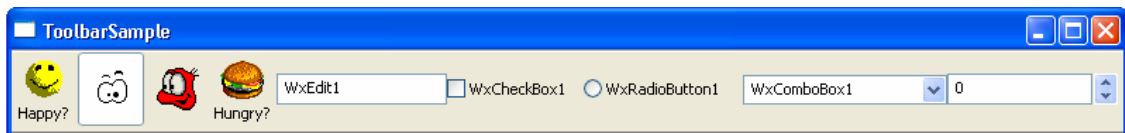


Figure 9.98 – The compiled toolbar sample

Advanced Topic – Reordering Toolbar Components

Occasionally you will layout the components on your toolbar only to think later I wish I had placed those buttons in a different order. (Of course this would not really happen because you would plan the GUI in advance).

wxDev-C++ allows you to select the components on the toolbar and reorganise them by dragging them into different positions. You can try it with the toolbar sample we have created. But you will notice at compile time that the components have not changed position. If you select the toolbar and right click, you can choose 'Change Creation Order' to reorder the components creation order.

Menu

The Menu palette contains all items related to menus. At present this is limited to Menu Bar and Popup Menu. The Menu bar appears along the top of a wxFrame, above the toolbar if there is one.

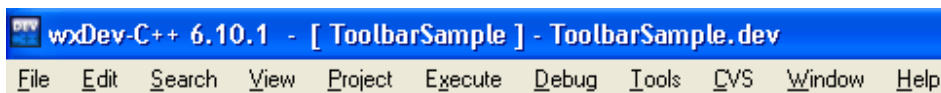


Figure 9.99 – The menu bar in wxDev-C++

A popup menu is usually connected to an event like a right mouse click when it will be made visible.

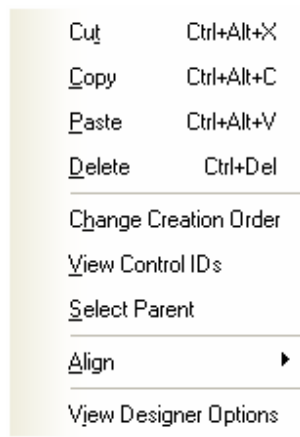


Figure 9.100 – A popup menu

To see how the menu designer works in wxDev-C++ we will create a sample project and add a menu bar and a popup menu to it.

- Open wxDev-C++
- Select File|New|Project...
- Select wxWidgets Frame
- Change Project Name to “MenuSample”
- Make sure C++ Project is selected
- Click the [OK] button
- Create a new folder in your projects
- Call the folder “MenuSample”
- Save the project in this folder
- Click the [Create] button

You will now have the usual blank frame.

Change the Component Palette to the Menu Palette. We will start by looking at the Menubar itself.

MenuBar

The behaviour of the MenuBar may seem a little strange the first time you use it. It is not like the other components which you can drop on the form and they look like the compiled component.

To test this we will drop a MenuBar on our sample project.

- Select a MenuBar
- Click anywhere on the form to place it.

You will find that instead of a bar at the top of the screen you get a funny little box like this.



Figure 9.101 – A MenuBar represented in the form designer

Now you might wonder what to do with this strange box. It won't let you resize it, double clicking on it does nothing either. The answer is in the properties.

Caption

This property is at present unused.

Menu Items

This is the important property, clicking on the button [...] that appears in the Property Inspector produces the Menu Item Editor.

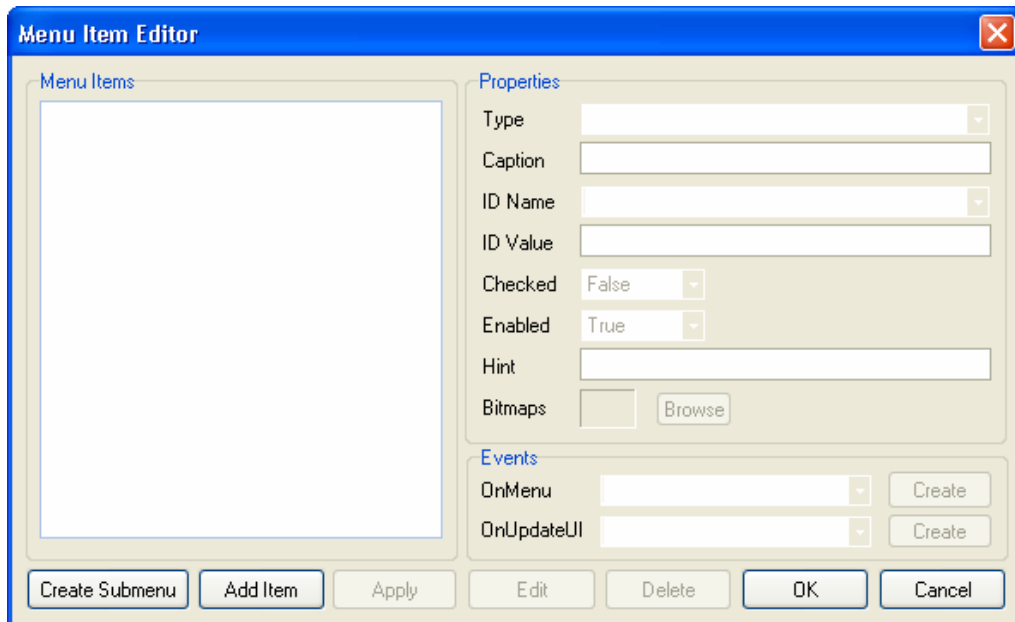


Figure 9.102 – The Menu Item Editor

The Menu Item Editor allows you to create new menu entries. It also allows you access to the various menu properties. Some of these are the same as the general item properties, like ID Name and ID Value. We will consider the other properties.

Type

<u>Menu Item Type</u>	<u>Purpose</u>
Menu Item	This menu item will be a normal menu item
Seperator	This menu item will be drawn as a line to

	separate different parts of the menu
Check Item	This item will be created with a check box
Radio Item	This item will be created as part of a radio group
File History	The file history item makes it easier to implement a MRU (Most Recently Used) file list

Caption

This string property sets the text that will be shown for this item. If any letter in this string is preceded by a '&' ampersand then this letter will be used as the menu options mnemonic.

Additionally the caption can include a keyboard accelerator. This is achieved by appending '\t' to the end of the caption followed by a command key 'CTRL', 'SHIFT' or 'ALT' then a '-' or '+' sign followed by the accelerator key. See Chapter 11 Mnemonics and Keyboard Accelerators for more details.

Checked

This property determines if a checkable menu item is created in a checked or unchecked state.

Enabled

This property determines if the menu item is enabled or disabled upon creation.

Hint

This string property allows you to set a hint that describes the menu's purpose. By default this string will be displayed in a status bar, if one exists.

Bitmaps

The Bitmap property allows you to specify a bitmap that will be displayed alongside the menu item. Usually this helps to clarify its purpose.

Using the Menu Editor

When you first open the Menu Editor you will notice that most of the controls are disabled. The 'Create Submenu' button doesn't do anything at this stage, since you need a root menu item to add the submenu to. The OK button will close the dialog saving the

changes you have made and the Cancel button will close the dialog without saving any of the changes you have made.

To start therefore, you need to use the Add Item button. We will try this to see how it works.

If you don't have the Menu Editor open, then open it
Click the [Add Item] button

You will notice that the tree control to the left now has a single item in it. This has a default name of MenuItem1. This is a root menu item, this means that it will appear along the top of the menu bar.

Change the Caption to be '&File'
Then click in the box ID Name

You should see that the default ID name value is created for you. (Or you could use the drop down box to select a predefined special meaning ID. We shall stick with the defaults.) We will leave the most of the other values at their defaults.

Change the Hint property to read 'This is the root file menu'
Click the [Apply] button.

Now we shall add another root level item.

Click the button [Add Item]
Change the Caption to be '&Help'
Click in the ID Name box
Change the Hint property to read 'This is the root help menu'
Click the [Apply] button

We now have two root items like this.

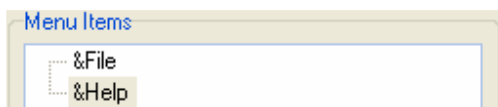


Figure 9.103 – The two root menu items

Now we want to add children to the root items.

Select the File root by clicking on it.
Now click on the button [Create Submenu]
Change the Caption to 'Item with Picture'
Click in the ID Name box
Change the Hint property to read 'This is a picture menu item'
Click on the [Browse] button next to bitmaps
Click on the [Load] button

Browse to the Icons folder in the wxDev-C++ install folder
Select the 'Danger' icon
Click the [Apply] button

You will now have three items the second one is indented, this is because it is a child of the item above. Since it is indented one level it will appear as the first item on the File drop down menu.

Select the File root by clicking on it.
Now click on the button [Create Submenu]
Change the Caption to 'Disabled Item'
Click in the ID Name box
Change the Hint property to read 'This is a disabled menu item'
Click the [Apply] button

Select the File root by clicking on it.
Now click on the button [Create Submenu]
Change the Caption to 'Parent Item'
Click in the ID Name box
Change the Hint property to read 'This menu item has a child'
Click the [Apply] button

The last item we created will be used to hold a child menu.

Select the item 'Parent Item' by clicking on it
Now click on the button [Create Submenu]
Change the Caption to 'Child Item'
Click in the ID Name box
Change the Hint property to read 'This menu item is a child'
Click the [Apply] button

Now we will create other types of menu.

Select the item '&Help' by clicking on it
Now click on the button [Create Submenu]
Change the Type property to 'Checked Item'
Change the Caption to 'Checked Item'
Click in the ID Name box
Changed the Checked property to True
Change the Hint property to read 'This is a checked item'
Click the [Apply] button

Select the item '&Help' by clicking on it
Now click on the button [Create Submenu]
Change the Type property to 'Seperator'
Click the [Apply] button

Select the item '&Help' by clicking on it
 Now click on the button [Create Submenu]
 Change the Type property to 'Radio Item'
 Change the Caption to 'First Radio Item'
 Click in the ID Name box
 Changed the Checked property to True
 Change the Hint property to read 'This is a radio item'
 Click the [Apply] button

Select the item '&Help' by clicking on it
 Now click on the button [Create Submenu]
 Change the Type property to 'Radio Item'
 Change the Caption to 'Second Radio Item'
 Click in the ID Name box
 Changed the Checked property to False
 Change the Hint property to read 'This is a radio item'
 Click the [Apply] button
 Click the [OK] button

The list of menu items should now look like this

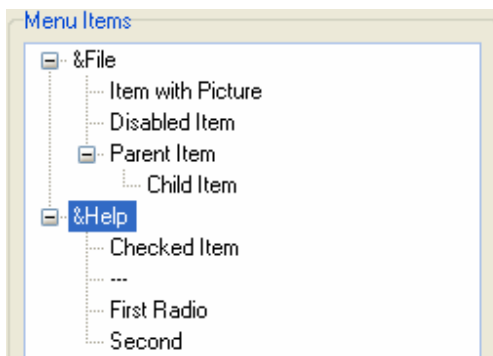


Figure 9.104 – The completed menu

Drop a panel component on the form
 Finally drop a status bar on the form
 Now press <F9> to compile and run



Figure 9.105 – The completed menu at compile time

We will finish the sample by looking at the popup menu.

PopupMenu

A Popup menu is a menu that appears under certain circumstances. For example right click on wxDev-C++ and a menu will appear. This is a Popup menu. You can design the menu in the form designer, but to show it you need to use some code.

- Select a PopupMenu from the component palette
- Drop it on the design form

Again you will see a little square. When you select the square you will see that the Popup menu has the same properties as a Menubar.

- Select the PopupMenu
- Click on the property 'Edit Menu Items'
- Now create two items
- Call the first 'Parent Item'
- Call the second 'Item two'
- Click the [Apply] button after creating each one
- Select 'Parent Item'
- Click the [Create Submenu] button
- Call this item 'Child Item'
- Click the Apply button
- Click the [OK] button

Now we need to be able to show the Popup menu.

- Select a Button control from the component palette
- Click on the designer form to place it
- Select the Event Tab
- In the OnClick event drop down the box and select <Add New Function>
- Underneath the line // insert your code here add the line

```
PopupMenu( WxPopupMenu1, WxButton1->GetRect().x, WxButton1->GetRect().GetBottom() );
```

- Press <F9> to compile and run

Now try pressing the button to see your newly created menu. Try playing with the menu item editor to add new items of different types.

It is also possible to rearrange items by dragging and dropping them in the tree list. Just dragging an item will place it after the item you drag it to. If you hold down the Shift key it will place it as a child of the item you drag it to.

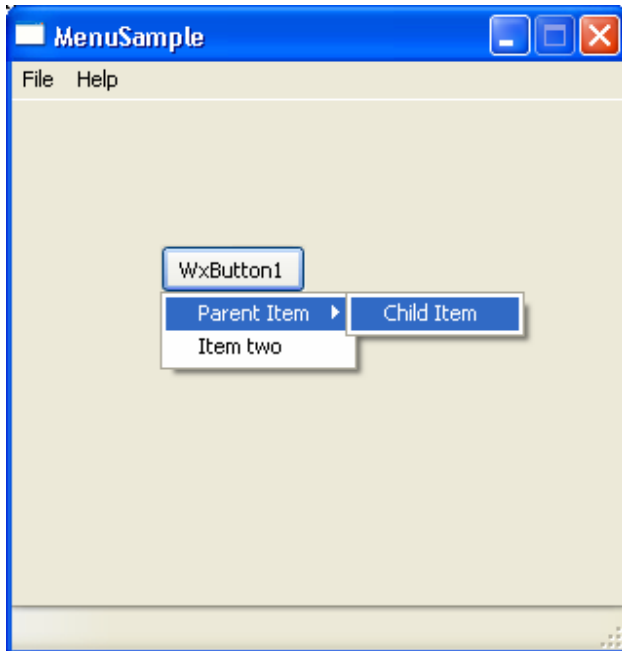


Figure 9.106 – The completed menu sample

Dialogs

Last chapter we looked at creating dialogs. wxDev-C++ comes with a number of standard dialogs that are ready for you to use. These include file opening/saving dialogs, printing dialogs and message dialogs. This section takes a look at each of these. The next chapter will cover how to actually display and retrieve values from some of these dialogs.

Open File Dialog

The `OpenFileDialog` is designed to allow the user to choose a file to open. The programmer may set a default location and file name or limit the user to files by a certain name.

The `Open` and `SaveFileDialogs` in wxDev-C++ are both `wxFileDialogs` then only difference is that the first is created with the `wxFD_OPEN` flag set the second with `wxFD_SAVE` set. This distinction is hidden from the user though.

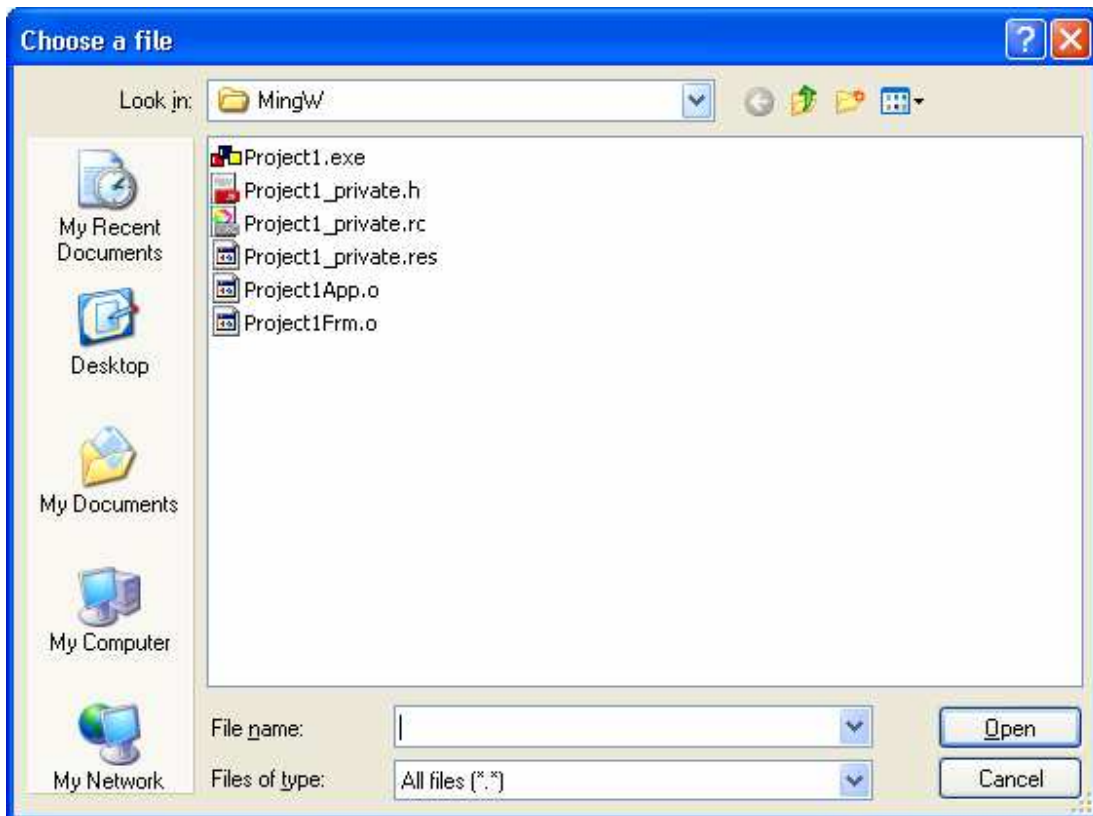


Figure 9.107 – The OpenFileDialog (wxFileDialog)

Default Dir

This string property allows you to set a default directory for the dialog to open at. For example:

```
C:/Program Files/
```

Default File

This string property contains a name of the file you want to select by default. For example to create a dialog with the file 'Default.txt' preselected you would use this line.

```
Default.txt
```

Extensions

This string property allows the user to specify which types of file can be selected. Several extensions can be registered separated by a '|'. The format to register an extension is <Description>|*.<Extension>. For example to create an file dialog that only allows the user to select .htm or .txt files you would use this line.

HTML files (*.htm)|*.htm|*.html|Text files (*.txt)|*.txt

File Dialog Style

This control uses the following flags to define its style

Flag Name	Purpose
wxCHANGE_DIR	If this flag is set the programs current working directory will be changed to the directory the user has chosen to open the file from
wxFILE_MUST_EXIST	Only allows existing file to be chosen
wxHIDE_READONLY	Hides read only files
wxMULTIPLE	Allows the user to select multiple items to open

Message

This string property allows you to set the message the user will see on the top of the dialog.

Save File Dialog

Much of the information for the OpenFileDialog applies to the SaveFileDialog. Only the differences will be covered here.

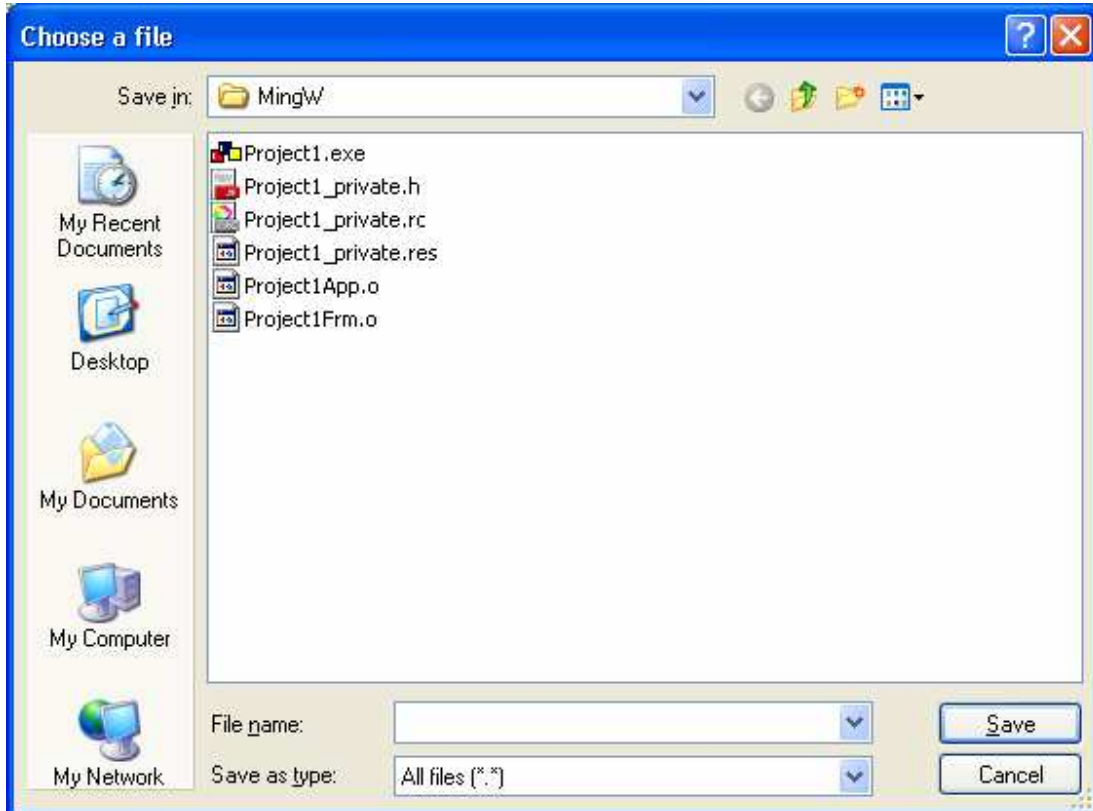


Figure 9.108 – The SaveFileDialog (wxFileDialog)

In fact the only difference is in the style flags.

File Dialog Style

This control uses the following flags to define its style

Flag Name	Purpose
wxCHANGE_DIR	If this flag is set the programs current working directory will be changed to the directory the user has chosen to save the file to
wxHIDE_READONLY	Hides read only files.
wxOVERWRITE_PROMPT	If this flag is set then a dialog will be displayed if the file already exists to alert the user.

Progress Dialog

The wxProgressDialog allows you to display a dialog with a message and a progress bar. This is usually used during lengthy operations, such a saving or modifying large files, or carrying out intensive calculations. It is possible to display other information such as the time the operation has taken and an estimate of when it will complete.

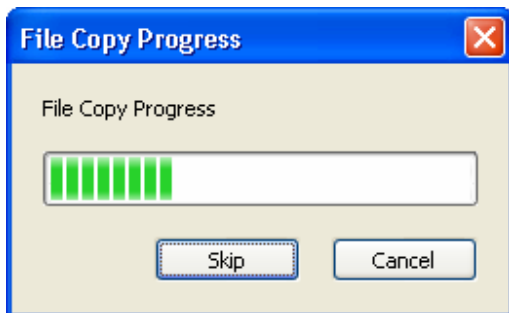


Figure 9.109 – The Progress Dialog

Auto Show

If this Boolean property is set to true the dialog will disappear as soon as the guage reaches the end.

MAX Value

This integer property allows you to set the maximum value the guage can reach.

Message

This string property allows you to set the message that appears on the body of the dialog.

Progress Dialog Style

This control uses the following flags to define its style

Flag Name	Purpose
wxPD_APP_MODAL	Stops the user from interacting with any part of the application while the dialog is shown, not just the parent window
wxPD_AUTO_HIDE	Causes the dialog to disappear once the gauge reaches its maximum value
wxPD_CAN_ABORT	Displays a Cancel button on the dialog
wxPD_CAN_SKIP	Displays a Skip button on the dialog
wxPD_ELAPSED_TIME	Causes the dialog to show how much time has elapsed since the dialog was created
wxPD_ESTIMATED_TIME	Causes the dialog to display much much time the operation is estimated to take
wxPD_REMAINING_TIME	Causes the dialog to show how much time is left
wxPD_SMOOTH	Causes the gauge to progress smoothly

Title

This string property allows you to set the message that appears on the caption of the dialog.

Colour Dialog

The wxColourDialog is used to offer your users a means to select different colours.



Figure 9.110 – The ColourDialog

The ColourDialog has no extra properties.

Dir Dialog

The wxDirDialog is similar to the file dialog, except instead of allowing the user to select a file, it allows them to select a directory.



Figure 9.111 – The DirDialog

Default Dir

This string property allows you to set a default directory for the dialog to open at. For example:

C:/Program Files/

Dir Dialog Style

This control uses the following flags to define its style

Flag Name	Purpose
wxDD_NEW_DIR_BUTTON	Causes a create new directory button to be displayed on the dialog

Message

This string property allows you to set the message the user will see on the top of the dialog.

Find Replace Dialog

The wxFindReplaceDialog is designed to allow the user to search for or replace words in a text based control. Unlike other dialogs it must have a parent since it uses this for searching. It also needs to be shown non-modally.

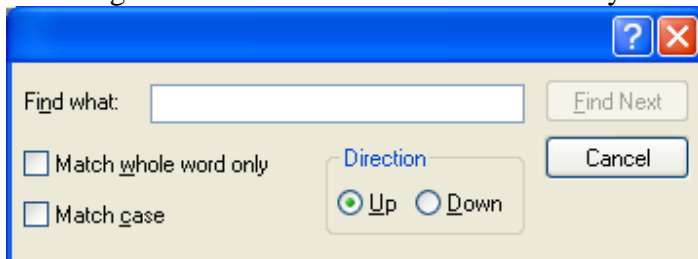


Figure 9.112 – The wxFindReplaceDialog in find mode

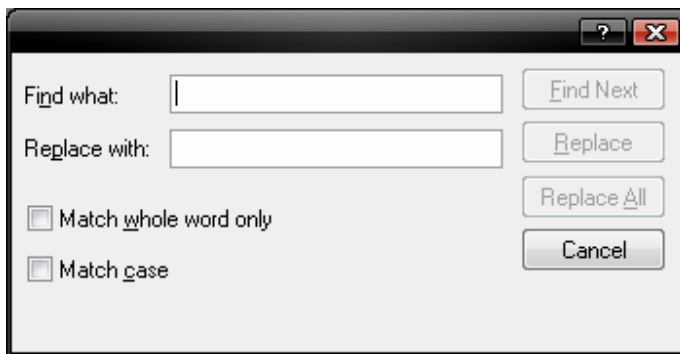


Figure 9.113 – The same dialog in replace mode

FindString

This string property allows you to set the string that you are searching for.

Flags

This control uses the following flags to control its operation

Flag Name	Purpose
wxFR_DOWN	If true start searching from top down. If false search from bottom up
wxFR_MATCHCASE	If true make the search case sensitive. If false make search case insensitive
wxFR_WHOLEWORD	If true match to whole words only, otherwise match to partial words

Styles

This control uses the following flags to define its style

Flag Name	Purpose
wxFR_NOMATCHCASE	This stop the user from allowing case sensitive matching
wxFR_NOUPDOWN	This stops the user from altering up down searching
wxFR_NOWHOLEWORD	This stops the user from changing wholeword searching
wxFR_REPLACEDIALOG	This alters whether the find or find replace dialog is shown

ReplaceString

This string property allows you to set the string that will be used as a replacement.

Title

This string property allows you to set the message that appears on the caption of the dialog.

Font Dialog

The font dialog allows the user to select a font from those installed on their system. It also allows them to alter various attributes such as font style, size and colour.

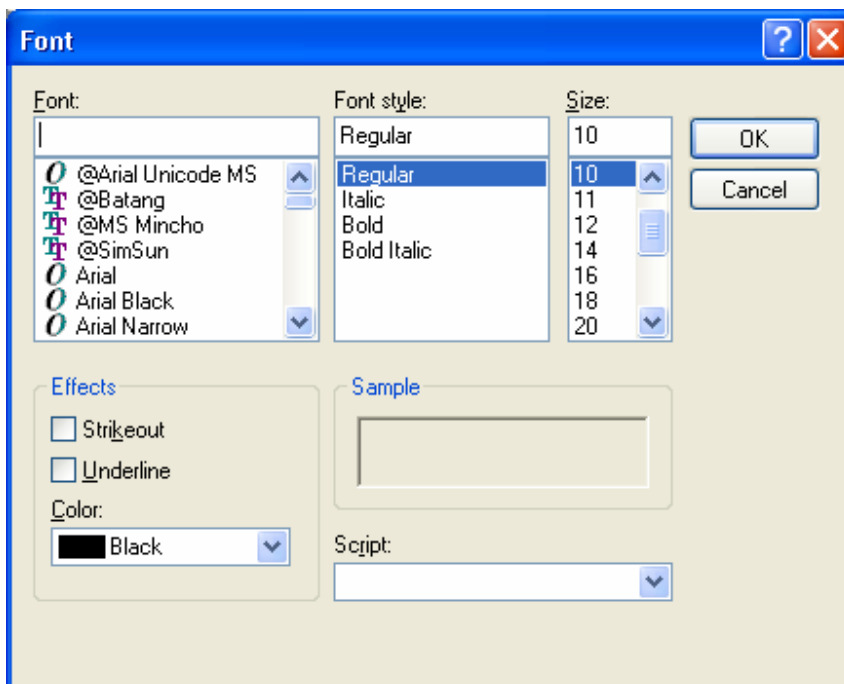


Figure 9.114 – The wxFontDialog

The Font Dialog has no extra styles.

Page Setup Dialog

The wxPageSetupDialog displays a dialog that allows the user to alter print settings such as paper size and orientation.

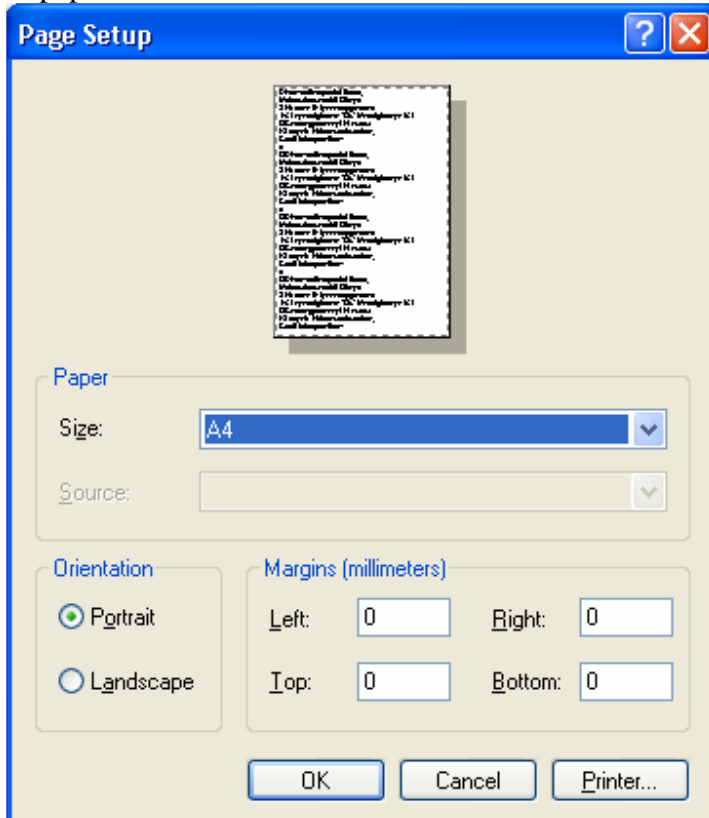


Figure 9.115 – The wxPageSetupDialog

MarginBottomRightX

This integer property allows you to set the default right hand margin size.

MarginBottomRightY

This integer property allows you to set the default bottom margin size.

MarginTopLeftX

This integer property allows you to set the default left hand margin size.

MarginTopLeftY

This integer property allows you to set the default top margin size.

MinMarginBottomRightX

This integer property allows you to set the minimum size right hand margin the user can choose.

MinMarginBottomRightY

This integer property allows you to set the minimum size bottom margin the user can choose.

MinMarginTopLeftX

This integer property allows you to set the minimum size left hand margin the user can choose.

MinMarginTopLeftY

This integer property sets the minimum size top margin the user can choose.

PaperId

This drop down property contains a long list of standard paper sizes that you can use to set the default shown.

Print Dialog

The wxPrintDialog displays the standard print dialog. This allows you to collect the necessary information to send to a printing process.

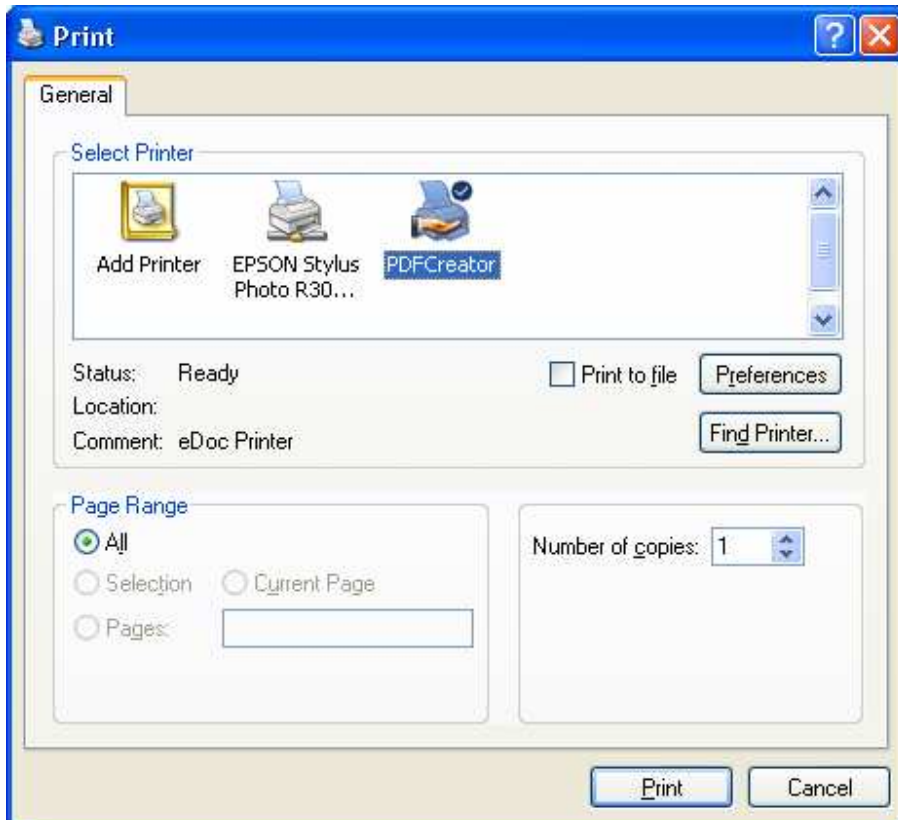


Figure 9.116 – The wxPrintDialog

From Page

This integer property allows you to set the page number to start printing from.

Max Page

This integer property allows you to set the maximum page number.

Min Page

This integer property allows you to set the minimum page number.

NumberOfCopies

This integer property allows you to set the number of copies to print.

PrintToFile

This boolean property allows you to set whether the “Print to file” check box is checked or not.

Selection

This boolean property allows you to set whether the “Selection” radio button is checked or not.

To Page

This integer property allows you to set the page number to print to.

Message Dialog

The wxMessageDialog allows you to display a message for your users. It allows a certain amount of customisation of the buttons and icon displayed.



Figure 9.117 – The wxMessageDialog

Caption

This string property allows you to set the message that appears on the caption of the dialog.

Message

This string property allows you to set the message that appears on the body of the dialog.

Message Dialog Style

This control uses the following flags to define its style

Flag Name	Purpose
wxCANCEL	Show a Cancel button
wxCENTRE	Centre the message on the dialog
wxICON_ERROR	Show an error icon
wxICON_EXCLAMATION	Show an exclamation mark icon
wxICON_HAND	Same as wxICON_ERROR
wxICON_INFORMATION	Show a information (i) mark icon
wxICON_QUESTION	Show a question mark icon
wxNO_DEFAULT	Use with the wxYES_NO flag to make the No button selected by default
wxOK	Show an OK button
wxYES_DEFAULT	Use with wxYES_NO flag to make the Yes button selected by default
wxYES_NO	Show Yes and No buttons

Text Entry Dialog

The wxTextEntryDialog allows the user to input a single line, it could be used as a login box, or to gather simple information.

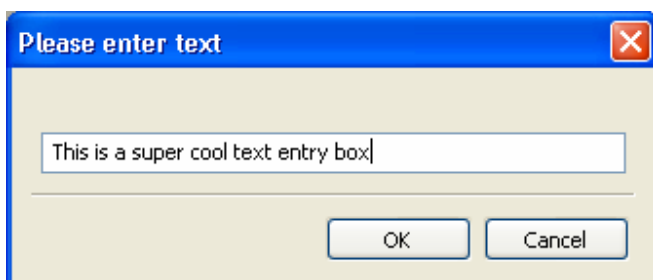


Figure 9.118 – The Text Entry Dialog

Caption

This string property allows you to set the message that appears on the caption of the dialog.

Dialog Style

This control uses the following flags to define its style

Flag Name	Purpose
wxCANCEL	Show a Cancel button
wxCENTRE	Centre the message on the dialog
wxNO_DEFAULT	Use with the wxYES_NO flag to make the No button selected by default
wxOK	Show an OK button
wxYES_DEFAULT	Use with wxYES_NO flag to make the Yes button selected by default
wxYES_NO	Show Yes and No buttons

Edit Style

This control uses the following flag to define the action of the edit control

Flag Name	Purpose
wxTE_PASSWORD	Display the users text entry as a line of '*' asterisks

Message

This string property allows you to set the message that appears on the body of the dialog.

Value

This string property allows you to set the default contents of the edit control.

Password Entry Dialog

This control is the same as the wxTextEntryDialog since it derives from it. The only difference is that the users entry is displayed as a line of asterisks by default.

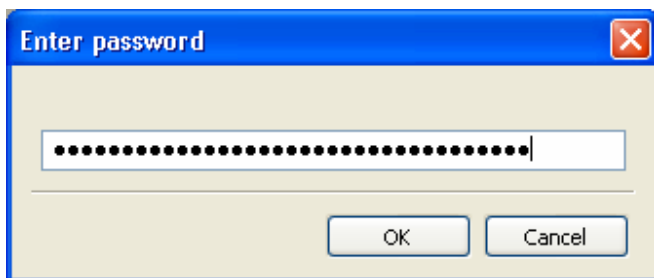


Figure 9.119 – The Password Entry Dialog

Caption

This string property allows you to set the message that appears on the caption of the dialog.

Dialog Style

This control uses the following flags to define its style

Flag Name	Purpose
wxCANCEL	Show a Cancel button
wxCENTRE	Centre the message on the dialog
wxNO_DEFAULT	Use with the wxYES_NO flag to make the No button selected by default
wxOK	Show an OK button
wxYES_DEFAULT	Use with wxYES_NO flag to make the Yes button selected by default
wxYES_NO	Show Yes and No buttons

Message

This string property allows you to set the message that appears on the body of the dialog.

Value

This string property allows you to set the default contents of the edit control.

Single Choice Dialog

The wxSingleChoiceDialog presents a list of choices to the user and allows them to select just one.

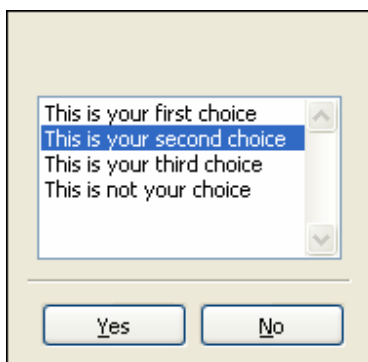


Figure 9.120 – The Single Choice Dialog

Caption

This string property allows you to set the message that appears on the caption of the dialog.

Items

This string list property allows you to set the list of items that the user can choose from.

Message

This string property allows you to set the message that appears on the body of the dialog.

Message Dialog Style

This control uses the following flags to define its style

Flag Name	Purpose
wxCANCEL	Show a Cancel button
wxCENTRE	Centre the message on the dialog
wxNO_DEFAULT	Use with the wxYES_NO flag to make the No button selected by default
wxOK	Show an OK button
wxYES_DEFAULT	Use with wxYES_NO flag to make the Yes button selected by default
wxYES_NO	Show Yes and No buttons

Multi Choice Dialog

The wxMultiChoiceDialog is the same as the Single Choice Dialog with the exception that it allows the user to make multiple choices.

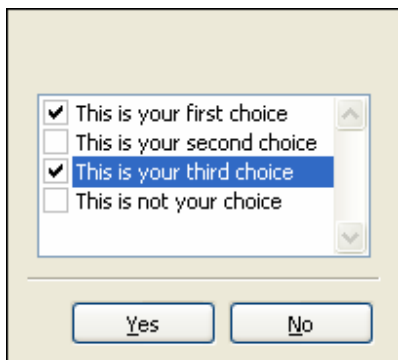


Figure 9.121 – The Multi Choice Dialog

Caption

This string property allows you to set the message that appears on the caption of the dialog.

Items

This string list property allows you to set the list of items that the user can choose from.

Message

This string property allows you to set the message that appears on the body of the dialog.

Message Dialog Style

This control uses the following flags to define its style

Flag Name	Purpose
wxCANCEL	Show a Cancel button
wxCENTRE	Centre the message on the dialog
wxNO_DEFAULT	Use with the wxYES_NO flag to make the No button selected by default
wxOK	Show an OK button
wxYES_DEFAULT	Use with wxYES_NO flag to make the Yes button selected by default
wxYES_NO	Show Yes and No buttons

Html Easy Printing

The wxHtmlEasyPrinting class is not a dialog in the same manner as the others here. It provides a class that makes printing HTML documents extremely simple. We will use this class in the sample application to show its power.

Footer Page

This integer property allows you to set which pages the footer string appears on.

Value	Corresponds to Flag	Purpose
0	wxPAGE_ODD	Footer appears on odd numbered pages
1	wxPAGE_EVEN	Footer appears on even numbered pages
2	wxPAGE_ALL	Footer appears on all pages

Footer String

This string property allows you to set an HTML string that is displayed on the foot of the pages. The string may contain the following macros.

Macro	Purpose
@DATE@	Inserts the current date in default format
@PAGENUM@	Inserts the page number

@PAGESCNT@	Inserts the total number of pages
@TIME@	Inserts the current time in default format
@TITLE@	Inserts the document title

Header Page

This integer property allows you to set which pages the footer string appears on. The values are the same as for the header page.

Header String

This string property allows you to set an HTML string that is displayed on the head of the pages. The string may contain the same macros as the footer.

Title

This string property allows you to set the name of the printing object. This will be used on the print preview and setup dialogs.

System

Timer

The wxTimer allows you to execute code at predefined intervals. The timer causes an event to be triggered whenever its time interval runs out.

AutoStart

If this property is set to true then the timer will begin to work as soon as it is created.

Interval

This property is an integer value that determines the interval between timer events in milliseconds.

Dial Up Manager

The wxDialUpManager class is designed to enable applications to interact with the Internet. Via the Dial Up Manager it is possible to check if the computer is online or to ask it to connect to the Internet. It is also possible to check events such as the computer connecting or disconnecting from the Internet.

MMedia

Media Ctrl

The wxMediaCtrl allows you to play various types of media such as video or music using the native control used on your platform for achieving this.



Figure 9.122 – The wxMediaCtrl before loading and running a file

The Media Control has the following unique property.

File Name

This string property contains the filename of the media file you want to play. Don't forget that the filename should be relative to the executable file or else the program will not be able to find it on a different users system.

Unofficial

TreeListCtrl

The wxTreeListCtrl is a mixture of a tree control and a list control. The first column contains the tree, the other columns contain lists. An example of this might be a file directory displayed as a tree. For each item in the tree there could be a corresponding list entry with file size, creation time, etc.

If you are using this control you will need to manually add the library libwxmsw28_treelistctrl.a to the list of libraries the linker looks for.

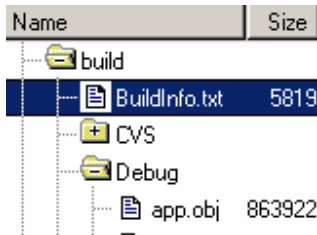


Figure 9.123 – The wxTreeListCtrl

The Tree List Control contains the following unique properties.

Columns

This property works exactly like the editor for the List Control. For more information look at the Columns property under List Control.

TreeList Styles

This control uses the following flags to define its style

Flag Name	Purpose
wxTR_COLUMN_LINES	This flag does not exist and will be ignored by the IDE
wxTR_DEFAULT_STYLE	Sets flags to the platforms native style
wxTR_EDIT_LABELS	This allows the user to edit the labels
wxTR_EXTENDED	Allows disjoint items to be selected
wxTR_FULL_ROW_HIGHLIGHT	Draws the selection highlight across the whole control, (needs the wxTR_NO_LINES flag set on Windows)
wxTR_HAS_BUTTONS	Places '+' and '-' buttons by parent items in the control
wxTR_HAS_VARIABLE_ROW_HEIGHT	Each row is drawn the correct height to fit its item. Otherwise all rows are the same height
wxTR_HIDE_ROOT	Hides the root node
wxTR_LINES_AT_ROOT	Draws lines between root items, needs wxTR_HIDE_ROOT to be set and wxTR_NO_LINES not to be set
wxTR_MULTIPLE	Multiple items can be selected at the same time
wxTR_NO_BUTTONS	Draw this control with no buttons
wxTR_NO_LINES	Hides the vertical level connectors
wxTR_ROW_LINES	Draws each row in contrasting colours
wxTR_SHOW_ROOT_LABEL_ONLY	This flag does not exist and will be

wxTR_SINGLE	ignored by the IDE Only one item can be selected at a time
wxTR_TWIST_BUTTONS	If this flag and the wxTR_HAS_BUTTONS flag are set then the tree is drawn with Mac style twist buttons

Sample Application Part 2 – The GUI

In the last chapter we created the frames and dialogs needed to create our sample application. In the chapter we will complete the GUI design before we implement the code that gets everything working. We will start on the main frame then work our way through the Splash screen and dialogs.

The Main Frame

The Main frame will consist of a menu bar, toolbar, status bar, a side toolbar and a notebook control. Without anymore delay let us make a start.

Open the project you created in the last chapter called HTMLEdit.
Select the editor tab labelled HTMLEditFrm.wxform.
If this tab is not available open it via the Project Manager, 'Project' Tab.

Now we are going to create the toolbar.

Change the component selector to Toolbar.
Select the toolbar component and drop it onto the form.
Change the toolbar's Name property to 'tlbMain'.

Next we are going to add the buttons and other components to the toolbar.

Add the following items from the Toolbar component palette to the toolbar.
Change the properties to match those listed.
For the bitmaps use the ones in the Tango Icons 16x16 folder.

Type	Tool Button
Bitmap	document-new16x16.png
Name	'btnNewFile'
Tooltip	'Create a new document'
Type	Tool Button
Bitmap	document-open16x16.png
Name	btnOpenFile'

Tooltip	'Open an existing document'
Type	Tool Button
Bitmap Name	document-save16x16.png
Tooltip	'Save this document'
Type	Separator
Type	Tool Button
Bitmap Name	document-print16x16.png
Tooltip	'Print this document'
Type	Tool Button
Bitmap Name	document-print-preview16x16.png
Tooltip	'Preview this document'
Type	Separator
Type	Tool Button
Bitmap Name	edit-undo16x16.png
Tooltip	'Undo last change'
Type	Tool Button
Bitmap Name	edit-redo16x16.png
Tooltip	'Redo last change'
Type	Separator
Type	ComboBox
Items	'Heading 1' 'Heading 2' 'Heading 3' 'Heading 4' 'Heading 5' 'Heading 6'
Name	'cmbHeadings'
Text	'Heading 1'
Tooltip	'Alter the heading type'
Type	Tool Button
Bitmap Name	format-justify-left16x16.png
Tooltip	'Use inside a tag to left align text'
Type	Tool Button
Bitmap Name	format-justify-center16x16.png
Tooltip	'Use inside a tag to center align text'
Type	Tool Button
Bitmap	format-justify-right16x16.png

Name	'btnAlignRight'
Tooltip	'Use inside a tag to right align text'

You should something that looks like this in the editor

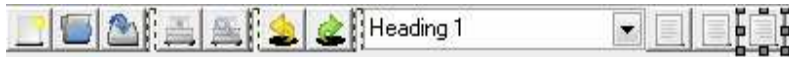


Figure 9.124 – The application toolbar in the design editor

The next component we will add is the status bar. This can be found on the Controls palette.

- Select a Status bar component and add it to the form.
- Change status bar's the Name property to 'stsInfo'.

Next we need a sizer component to hand the form the form's child components when it changes size. The sizers can be found on the Sizer palette.

- Select a BoxSizer component and add it to the form.
- Change the sizer's Name property to 'sizrMain'.
- Make sure the sizer's Orientation is set to wxHorizontal.

Now we need to create the side toolbar. Unfortunately we can't use the designer for this purpose so we need to 'fake' it. Instead we will use a Panel component as a parent for a series of buttons. We have a further problem in that some of the button labels are just text, the others are bitmaps. We need to use ordinary buttons and bitmap buttons to achieve this.

- Select the sizer you just dropped on the form.
- Select a Panel from the palette, you will find this under Window.
- Drop the panel onto the sizer.
- Change the Name property to 'pnlSideBar'.
- Set the alignment wxEXPAND to true and make sure the other alignment options are all set to false.
- Set Border property to 0.
- Set the panel's Height to 278 and its Width to 36.

Now we want to add the buttons, these can be found on the Controls palette. Add the controls listed in the following table altering the properties to match.

Type	Button
Font	Arial, Bold, size 10
Height	30
Label	'B'
Left	3
Name	'btnEmboldenText'
Tooltip	'Bold text style'

Top	0
Width	30
Type	Button
Font	Times New Roman, Italic, size 11
Height	30
Label	'I'
Left	3
Name	'btnItaliciseText'
Tooltip	'Italic text style'
Top	29
Width	30
Type	Button
Font	Arial, Regular, size 12, Underline
Height	30
Label	'U'
Left	3
Name	'btnUnderlineText'
Tooltip	'Underlined text style'
Top	58
Width	30
Type	Button
Font	Times New Roman, Italic, size 12
Height	30
Label	'F'
Left	3
Name	'btnChangeFont'
Tooltip	'Change font style'
Top	87
Width	30
Type	Button
Font	Courier, Regular, size 15
Height	30
Label	'-'
Left	3
Name	'btnInsertHorizontalLine'
Tooltip	'Insert a horizontal line'
Top	116
Width	30
Type	BitmapButton
Bitmap	Anchor.xpm (From the 'Project Images' Zip File)
Button Style	wxBU_AUTODRAW
Height	30
Left	3
Name	'btnCreateAnchorPoint'
Tooltip	'Create an anchor point'

Top	145
Width	30
Type	BitmapButton
Bitmap	Link.xpm (From the 'Project Images' Zip File)
Button Style	wxBU_AUTODRAW
Height	30
Left	3
Name	'btnCreateHyperlink'
Tooltip	'Create a hyperlink'
Top	174
Width	30
Type	BitmapButton
Bitmap	Table.xpm (From the 'Project Images' Zip File)
Button Style	wxBU_AUTODRAW
Height	30
Left	3
Name	'btnInsertTable'
Tooltip	'Insert a table'
Top	203
Width	30
Type	BitmapButton
Bitmap	image-x-generic16x16.png
Button Style	wxBU_AUTODRAW
Height	30
Left	3
Name	'btnInsertImage'
Tooltip	'Insert an image'
Top	232
Width	30

Now we are going to add the notebook component which will hold two pages. One of these will hold a preview pane and an edit pane. The other will hold a preview pane only.

Select the sizer `sizerMain` on the form.
Switch to the Window palette and select `NoteBook`.
Drop this into the selected sizer.
Change the Name property to `'nbkEditor'`.
Change the Alignment flag `wxEXPAND` to true, make sure the others are set to false.
Set the Border property to 0.
Set the Height to 275 and the Width to 460.
Finally set the Stretch Factor to 1.

Now we need to add the pages. These can also be found on the Window palette.

Select the Notebook `nbkEditor`.

Select a NotebookPage component and drop it on the notebook.
Change the Name property to 'nbpCombinedView'.
Change the Label property to 'Combined HTML Preview'.
Again select the Notebook nbkEditor.
Select another NotebookPage component and drop it on the notebook.
Select it by clicking on its tab, then clicking within its page.
Change the Name property to 'nbpPreview'.
Change the Label property to 'Preview'.

Having added the pages we need to edit the controls they contain.

Select the Combine HTML Preview page by clicking on its tab then clicking on the page.
From the Sizers palette select another BoxSizer.
Drop this on the page.
Change the sizer's Name property to 'sizrCombinedView'.
Make sure the sizer's orientation is set to wxHORIZONTAL.
Now from the Window palette select a SplitterWindow and drop on the sizer.
Make sure the splitter window's Alignment property is set to wxEXPAND only.
Change the splitter's Border property to 0.
Set the Height property to 248 and the Width property to 195.
Change the Name property to 'sptCombinedView'.
Make sure the Orientation property is set to wxVertical.
And set the Stretch Factor property to 1.
Next add an HTML Window from the same palette.
Set the Name property to 'htmCombinedPreview'.
Set the Height property to 89 and the Width property to 185.
From the Controls palette select a Memo control.
Add the control to the splitter window.
Name it 'mmoTextEditor'.
Set the Height to 139 and the Width to 185.
Use the Strings property to clear the text displayed.
Now switch to the Preview page by clicking on its tab.
Click on its page to select it.
Add a BoxSizer to this tab and name it 'sizrPreview'.
From the Window palette add a HTML Window to the sizer.
Set the HTML Windows Alignment to wxEXPAND only.
Set the Border to 0 and Stretch Factor to 1.
Change the Name to 'htmPreview'.

We are almost there for the Main Form. We just need to add the Menu and various standard dialogs. We will add the menu next.

From the Menu palette select the MenuBar control.
Drop this onto a free area on the Main Form.
Name it 'mnuMainBar'.

Then use the Menu Items property to bring up the Menu Editor and add the following items. The option 'Child of ..' means make it a sub menu of ... The option 'Root' means it should be added as a top level item.

Type	Menu Item
Location	Root
Caption	'&File'
ID Name	Accept Default
ID Value	Accept Default
Type	Menu Item
Location	Child of File
Caption	'&New\tCtrl+N'
ID Name	Accept Default
ID Value	Accept Default
Hint	'Create a new document'
Bitmap	document-new16x16.png
Type	Menu Item
Location	Child of File
Caption	'&Open\tCtrl+O'
ID Name	Accept Default
ID Value	Accept Default
Hint	'Open an existing document'
Bitmap	document-open16x16.png
Type	Menu Item
Location	Child of File
Caption	'&Save\tCtrl+S'
ID Name	Accept Default
ID Value	Accept Default
Hint	'Save current document'
Bitmap	document-save16x16.png
Type	Separator
Location	Child of File
ID Name	Accept Default
ID Value	Accept Default
Type	File History
Location	Child of File
ID Name	Accept Default
ID Value	Accept Default
Type	Menu Item
Location	Child of File
Caption	'Page Set&up'
ID Name	Accept Default
ID Value	Accept Default
Hint	'Set up the printer'
Bitmap	document-properties16x16.png

Type	Menu Item
Location	Child of File
Caption	'Print Pre&view'
ID Name	Accept Default
ID Value	Accept Default
Hint	'Preview the printed document'
Bitmap	document-print-preview16x16.png
Type	Menu Item
Location	Child of File
Caption	'&Print\tCtrl+P'
ID Name	Accept Default
ID Value	Accept Default
Hint	'Print current document'
Bitmap	document-print16x16.png
Type	Separator
Location	Child of File
ID Name	Accept Default
ID Value	Accept Default
Type	Menu Item
Location	Child of File
Caption	'E&xit'
ID Name	Change to wxID_EXIT
ID Value	Accept Default
Hint	'Exit this program'
Bitmap	process-stop16x16.png
Type	Menu Item
Location	Root
Caption	'&Edit'
ID Name	Accept Default
ID Value	Accept Default
Type	Menu Item
Location	Child of Edit
Caption	'&Undo\tCtrl+Z'
ID Name	Accept Default
ID Value	Accept Default
Hint	'Undo the last change'
Bitmap	edit-undo16x16.png
Type	Menu Item
Location	Child of Edit
Caption	'&Redo\tCtrl+Y'
ID Name	Accept Default
ID Value	Accept Default
Hint	'Redo the last undo'
Bitmap	edit-redo16x16.png
Type	Separator

Location	Child of Edit
ID Name	Accept Default
ID Value	Accept Default
Type	Menu Item
Location	Child of Edit
Caption	'Cu&t\tCtrl+X'
ID Name	Accept Default
ID Value	Accept Default
Hint	'Cut the select text'
Bitmap	edit-cut16x16.png
Type	Menu Item
Location	Child of Edit
Caption	'&Copy\tCtrl+C'
ID Name	Accept Default
ID Value	Accept Default
Hint	'Copy the select text'
Bitmap	edit-copy16x16.png
Type	Menu Item
Location	Child of Edit
Caption	'&Paste\tCtrl+V'
ID Name	Accept Default
ID Value	Accept Default
Hint	'Paste text from the clipboard'
Bitmap	edit-paste16x16.png
Type	Menu Item
Location	Root
Caption	&Help
ID Name	Accept Default
ID Value	Accept Default
Type	Menu Item
Location	Child of Help
Caption	'&About'
ID Name	Accept Default
ID Value	Accept Default
Hint	'Display the about box'
Bitmap	dialog-information16x16.png

If you are still here after that great long list be glad that you aren't writing an application like Microsoft Word. Any way take a little break and check to see that what you have in the menu editor looks something like the following figure.

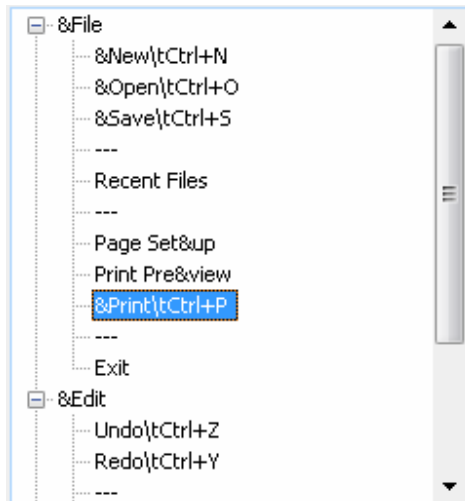


Figure 9.125 – The list of menu items in the Menu Editor

If all is ok then close the menu editor by pressing [OK].

We have just five more items to add and all of them are dialogs.

- Switch to the Dialog palette.
- Select an OpenFileDialog.
- Drop it on a clear area on the form.
- Change its Extensions property to read '*.htm;*.html'.
- Change the Message property to 'Choose a file'.
- Change its Name to 'dlgFileOpen'.
- Next select a SaveFileDialog.
- Drop on the form.
- Change the Extensions to '*.htm;*.html'.
- Change the Message to 'Choose a file'.
- Change its Name to 'dlgFileSave'.
- Select a FontDialog and add to the form.
- Change its name to 'dlgFont'.
- Select an HTMLEasyPrinting and add to the form.
- Change its Footer String to 'Printed Using Simple HTML Editor - @DATE@ - Page @PAGENUM@ of @PAGESCNT@ Pages'.
- Change its Header String to '@TITLE@'.
- Change its name to 'dlgPrint'.
- Change its title to 'Simple HTML Printing'.
- Finally add a MessageDialog.
- Change its Caption to 'Save Changes?'.
- Change its Message to 'The contents of this file has changed\n Do you want to save the changes?'.
- Set the Message Dialog Style to wxICON_EXCLAMATION, wxYES_DEFAULT and wxYES_NO only.
- Change its Name to 'dlgSaveChanges'.

Relax that is the main form complete. If you wish to see some reward for your hard work, compile and run the application. Try altering the size of the form to see how the components resize themselves to match. Try playing with the menus you should see as you hover over them that the message on the status bar changes to tell you the menus purpose.

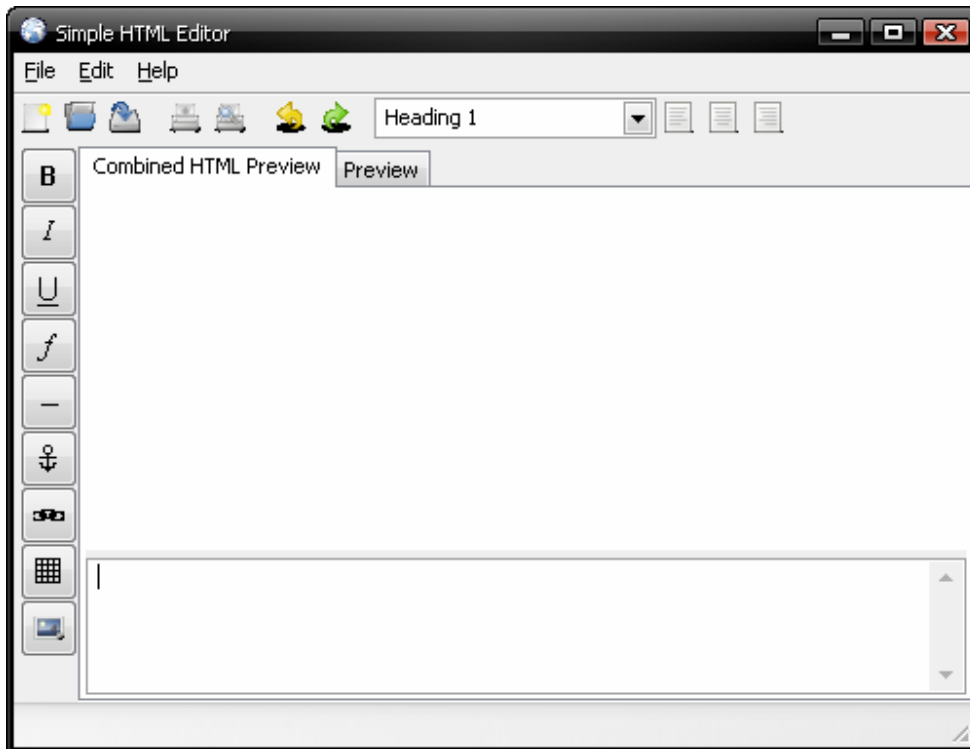


Figure 9.126 – Hopefully your form looks something like this.

When you have finished playing we are going to create the About box.

The About Box

We will start by adding the bitmap for our application.

Switch to the Controls palette and select a StaticBitmap.

Drop it on the form.

We want to change the Background Color and use the Standard Color wxWHITE.

Use the Picture property to select the 'Splash.png' from the 'Project Images' zip file.

Change the General Styles to wxSUNKEN_BORDER.

Change Height to 145 and Width to 249.

Change the Left to 4 and Top to 3.

Change the Name to 'bmpSplash'.

Next we want to add some labels to provide basic information to the user. We will surround this with a Static Box.

Add the following components to the form.

Type	staticbox
Caption	'About Simple HTML Editor'
Height	71
Left	3
Name	'stbInfoBox'
Top	150
Width	250
Type	StaticText
Height	17
Label	'Author:'
Left	9
Name	'txtAuthor'
Top	167
Width	38
Type	StaticText
Height	17
Label	'Project Site:'
Left	9
Name	'txtProjectSite'
Top	182
Width	64
Type	StaticText
Height	17
Label	'Icons From:'
Left	9
Name	'txtIconsFrom'
Top	198
Width	59
Type	StaticText
Height	17
Label	'Sof.T'
Left	77
Name	'txtAuthorsName'
Top	167
Width	30
Type	HyperLinkCtrl
Height	17
Label	'Programming with wxDev-C++'
Left	77
Name	'hypProjectSite'

Top	182
Width	141
Type	HyperLinkCtrl
Height	17
Label	'Tango Desktop Project'
Left	77
Name	hypIconsFrom
Top	198
Width	146
Type	Button
Height	28
ID Name	wxID_OK
Label	'Close'
Left	81
Name	btnClose
Top	224
Width	90

That is it for the About Box. You should now have something like the following image.



Figure 9.127 – The About Box in the form designer

Now we will move on to the simplest form in the project the Splash Screen. This will appear when we start the project while the main form loads.

The Splash Screen

For this form we just add a StaticBitmap. This is used for the Image and for cutting out the shape of the form.

Add a StaticBitmap.
 Set the Name to 'bmpSplashImage'.
 Set the Picture to 'Splash.png'.
 Set the Left to 0 and the Top to 0.
 Set the Height to 131 and the Width to 212.

That is it for the Splash Screen, all the magic takes place in the next chapter where we delve into the coding. Your form designer should look like this.



Figure 9.128 – The Splash Screen in the form designer

Next we will add the components to the InsertHyperlinkDlg.

The Insert Hyperlink Dialog

While not as simple as the Splash Screen, this dialog only needs 5 controls. Using the table add the following controls.

Type	staticbox
Caption	'Link Location'
Height	75
Left	11
Name	'stbLinkLocation'
Top	4
Width	342
Type	StaticText
Height	17
Label	'Type in a url to link to, or choose a local link from the dropdown box'
Left	18
Name	'txtLinkLocation'
Top	25
Width	322
Type	ComboBox
Height	21
Left	19
Name	'cmbHyperLink'

Text	“
Top	43
Width	322
Type	Button
Height	27
ID Name	wxID_OK
Label	‘OK’
Left	12
Name	‘btnOK’
Top	88
Width	91
Type	Button
Height	27
ID Name	wxID_CANCEL
Label	‘Cancel’
Left	250
Name	‘btnCancel’
Top	88
Width	101

At this point you should have something that looks like this in the form designer.

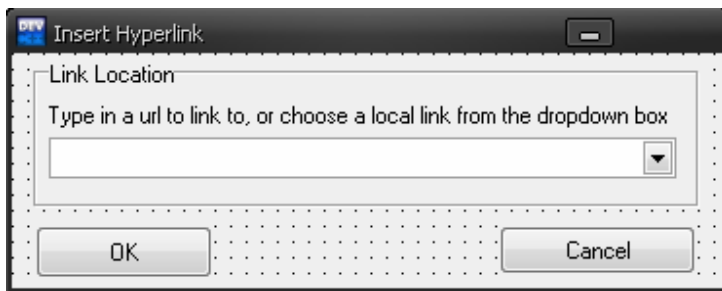


Figure 9.129 – The Insert Hyperlink Dialog

The next two dialogs are a little more complex. We shall look at the Create Table dialog next.

The Create Table Dialog

We need various different controls on this dialog. We need spin controls to alter the numeric properties such as numbers of rows and columns. We also need radio controls to allow us to set some exclusive properties on the table. Along with these we will use some of the other more common controls. The first thing we must do is drop a Panel onto the dialog. This is because the first panel added will be stretched to cover the dialog. If we miss this step we will be messed up later. Once again follow the table below.

Type	Panel
Name	‘pnlBackground’

Type	StaticText
Height	17
Label	'Rows'
Left	5
Name	'txtRows'
Top	5
Width	31
Type	StaticText
Height	17
Label	'Cell Spacing'
Left	103
Name	'txtCellSpacing'
Top	5
Width	63
Type	StaticText
Height	17
Label	'Cols'
Left	5
Name	'txtCols'
Top	29
Width	24
Type	StaticText
Height	17
Label	'Cell Padding'
Left	103
Name	'txtCellPadding'
Top	29
Width	63
Type	StaticText
Height	17
Label	'Border'
Left	5
Name	'txtBorder'
Top	53
Width	35
Type	StaticText
Height	17
Label	'Table Width'
Left	103
Name	'txtTableWidth'
Top	53
Width	62
Type	SpinCtrl
Height	22
Left	45

Maximum Value	100
Minimum Value	0
Name	'spnRows'
Top	5
Value	2
Width	53
Type	SpinCtrl
Height	22
Left	168
Maximum Value	100
Minimum Value	0
Name	'spnCellSpacing'
Top	5
Value	0
Width	55
Type	SpinCtrl
Height	22
Left	45
Maximum Value	100
Minimum Value	0
Name	'spnCols'
Top	29
Value	2
Width	53
Type	SpinCtrl
Height	22
Left	168
Maximum Value	100
Minimum Value	0
Name	'spnCellPadding'
Top	29
Value	0
Width	55
Type	SpinCtrl
Height	22
Left	45
Maximum Value	100
Minimum Value	0
Name	'spnBorder'
Top	53
Value	1
Width	53
Type	SpinCtrl
Height	22
Left	168

Maximum Value	100
Minimum Value	0
Name	'spnTableWidth'
Top	53
Value	0
Width	55
Type	RadioBox
Caption	'Table Alignment'
Height	43
Items	'Left' 'Center' 'Right'
Left	5
Major Dimension	1
Name	'rdbTableAlignment'
Radiobox Style	wxRA_SPECIFY_ROWS
Selected Button	0
Top	80
Width	218
Type	RadioBox
Caption	'Table Background'
Height	43
Items	'Use Colour' 'Use Image'
Left	5
Major Dimension	1
Name	'rdbTableBackground'
Radiobox Style	wxRA_SPECIFY_ROWS
Selected Button	0
Top	126
Width	218
Type	Panel
Height	20
Left	5
Name	'pnlColor'
General Styles	wxSIMPLE_BORDER
Top	174
Width	88
Type	Edit
Height	19
Left	98
Name	'edtImageSource'
Text	','
Top	174
Width	124

Type	Button
Height	26
Label	'Choose Colour'
Left	5
Name	'btnChooseColour'
Top	201
Width	88
Type	Button
Height	26
Label	'Choose Image'
Left	134
Name	'btnChooseImage'
Top	201
Width	88
Type	StaticLine
Left	5
Length	217
Name	'stlButtonSeparator'
Orientation	wxLI_HORIZONTAL
Top	233
Type	Button
Height	26
ID Name	wxID_OK
Label	'OK'
Left	5
Name	'btnOK'
Top	241
Width	75
Type	Button
Height	26
ID Name	wxID_CANCEL
Label	'Cancel'
Left	147
Name	'btnCancel'
Top	241
Width	75
Type	ColorDialog
Name	'dlgColorChooser'
Type	OpenFileDialog
Message	'Choose an image file'
Name	'dlgFileChooser'

The Create Table Dialog should look like this in the form designer.

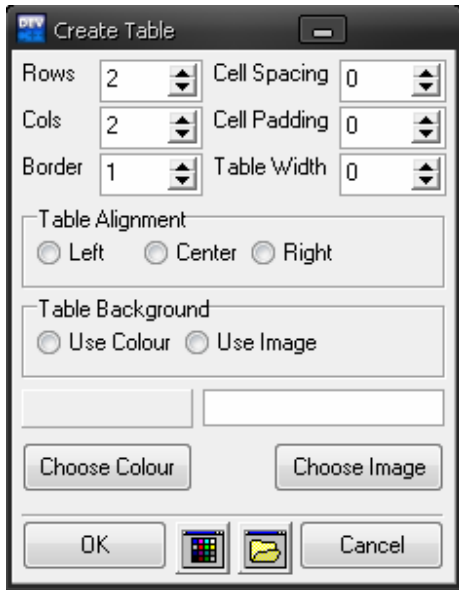


Figure 9.130 – The Create Table Dialog in the form designer

Relax once more and and take a break before the final dialog. In the final section we are going to add the controls to the Insert Image Dialog. This is slightly smaller than the last dialog.

The Insert Image Dialog

This dialog allows you to select an image and set its properties. As before add the controls as listed in the following table.

Type	StaticText
Height	17
Label	'Height'
Left	4
Name	'txtHeight'
Top	6
Width	35
Type	StaticText
Height	17
Label	'Width'
Left	4
Name	'txtWidth'
Top	38
Width	32
Type	StaticText
Height	17
Label	'Border'
Left	4

Name	'txtBorder'
Top	67
Width	35
Type	StaticText
Height	17
Label	'Image Source'
Left	4
Name	'txtImageSource'
Top	95
Width	70
Type	StaticText
Height	17
Label	'Alternative Text'
Left	4
Name	'txtAlternativeText'
Top	122
Width	78
Type	SpinCtrl
Height	22
Left	48
Maximum Value	3000
Minimum Value	0
Name	'spnHeight'
Top	6
Value	100
Width	63
Type	SpinCtrl
Height	22
Left	48
Maximum Value	3000
Minimum Value	0
Name	'spnWidth'
Top	38
Value	100
Width	63
Type	SpinCtrl
Height	22
Left	48
Maximum Value	3000
Minimum Value	0
Name	'spnBorder'
Top	67
Value	0
Width	63
Type	StaticBitmap

Height	84
Left	120
Name	'imgPreview'
Top	6
Width	102
Type	Edit
Height	19
Left	85
Name	'edtImageSource'
Text	''
Top	95
Width	83
Type	Edit
Height	19
Left	85
Name	'edtAlternativeText'
Text	''
Top	122
Width	136
Type	Button
Height	21
Label	'Browse'
Left	173
Name	'btnBrowseForImage'
Top	95
Width	49
Type	Button
Height	24
ID Name	'wxID_OK'
Label	'OK'
Left	13
Name	'btnOK'
Top	157
Width	72
Type	Button
Height	24
ID Name	'wxID_CANCEL'
Label	'Cancel'
Left	143
Name	'btnCancel'
Top	157
Width	72
Type	OpenFileDialog
Message	'Choose an image file'
Extensions	All Image Files *.bmp;*.gif;*.jpg;*.jpeg PNG files

Name	(*.png) *.png GIF files (*.gif) *.gif JPG files (*.jpg) *.jpg JPEG files (*.jpeg) *.jpeg 'dlgFileChooser'
-------------	--

That's it! Finally you have reached the end of the design of the forms. The Insert Image Dialog should look like the following image.

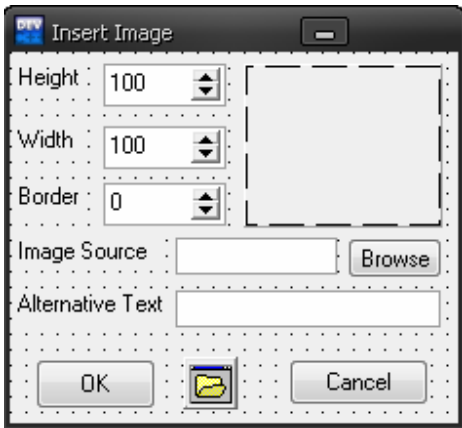


Figure 9.131 – The Insert Table Dialog in the form designer

Just to check that everything is all right compile and run the application. You will notice that at the moment you cannot see the other dialogs you have created. In the next chapter we will look at adding code to show the different dialogs. We will also add code to make all those buttons and menus do something useful.

Advanced Users

WARNING: You are advised to back up any files before altering them. This means you can go back and replace them if anything goes wrong.

I personally find the component palette to be a little awkward to work with. Either you have to show all files and scroll up and down looking for the ones you want or you have make a guess at which section houses which component. Wouldn't it be nice if you could decide what goes where, or have your own sections with your most used components? Well the good news is that you can, even better news it that there are two ways to do it.

1. To do this you need to use notepad. Open it then select File|Open. In the Open File dialog make sure the combo box labelled 'File of Type' is changed to read 'All Files'. Now browse to where you installed wxDev-C++. The file you are looking for is called 'devcpp.palette'.

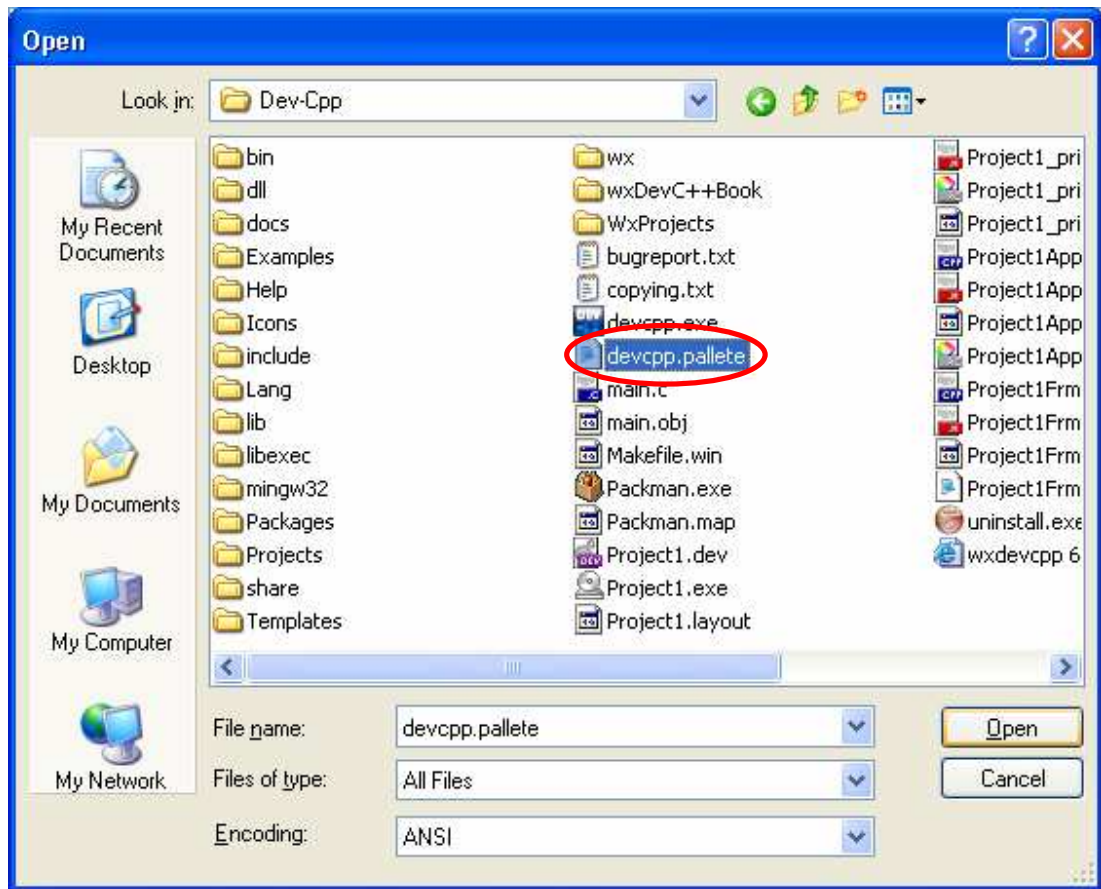


Figure 9.132 – Opening devcpp.pallete

You will be greeted by the contents of a file like this:

```
[Palette]
Sizers=TWxBoxSizer;TWxStaticBoxSizer;TWxGridSizer;TWxFlexGridSizer;
Controls=TWxStaticText;TWxButton;TWxBitmapButton;TWxToggleButton;TWxEdit;
TWxMemo;TWxCheckBox;TWxChoice;TWxRadioButton;TWxComboBox;TWxListBox;TWXListCtrl;TWxTreeCtrl;TWxGauge;TWxScrollBar;TWxSpinButton;TWxstaticbox;TWxRadioButton;TWxDatePickerCtrl;TWxSlider;TWxStaticLine;TWxStaticBitmap;TWxStatusBar;TWxChecklistbox;TWxSpinCtrl;
Window=TWxPanel;TWxNoteBook;TWxNoteBookPage;TWxGrid;TWxScrolledWindow;TWxHtmlWindow;TWxSplitterWindow;
Toolbar=TWxToolBar;TWxToolButton;TWxSeparator;TWxEdit;TWxCheckBox;TWxRadioButton;TWxComboBox;TWxSpinCtrl;
Menu=TWxMenuBar;TWxPopupMenu;
Dialogs=TWxOpenFileDialog;TWxSaveFileDialog;TWxProgressDialog;TWxColourDialog;TWxDirDialog;TWxFindReplaceDialog;TWxFontDialog;TWxPageSetupDialog;TWxPrintDialog;TWxMessageDialog;
System=TWxTimer;
[Version]
IniVersion=1
```

Each line contains one section on the palette. The start of the line before the '=' sign is the section name. The rest of the line is the list of components in that

section. Notice that each component is separated by a semi colon. The other thing to note is that each component name begins with a capital 'T'.

To create a new section, choose a name. Add this to the file by putting a new line in between 'System' and '[Version]'. Follow your name with an equals '=' sign. Then add the components you want in your section copy the names from those already in the file and separate each with a semi colon. Also end the line with a semi colon. For example

```
Favourites=TWxBoxSizer;TWxPanel;TWxEdit;TWxMemo;TWxMenuBar;TWxToolBar;TWx  
ToolButton;TWxStatusBar;
```

Save the file and then restart wxDev-C++ you should find your new section in the component palette.

2. The second method is to use the GUI tool Palette Manipulator available from the [Bonus download section](#) of the project site.

The first thing to do after running the program is to open the devcpp.pallete file. To do this go to File|Open. The following browse dialog will be displayed. Browse to the Dev-Cpp folder.

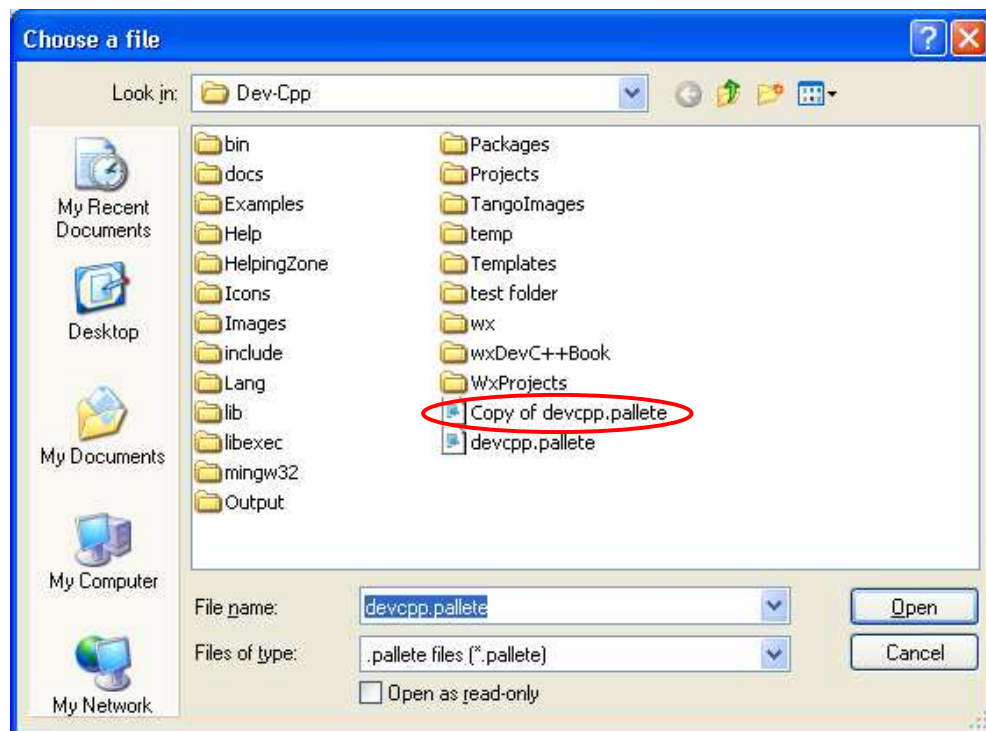


Figure 9.133 – Browsing for the devcpp.pallete file

Open this file. The left-hand pane will be filled with the names of the available palettes. Select any of these palettes and the centre pane will be filled with the

names of the components contained on this palette. The right-hand pane contains the names of the components available to add to this palette.

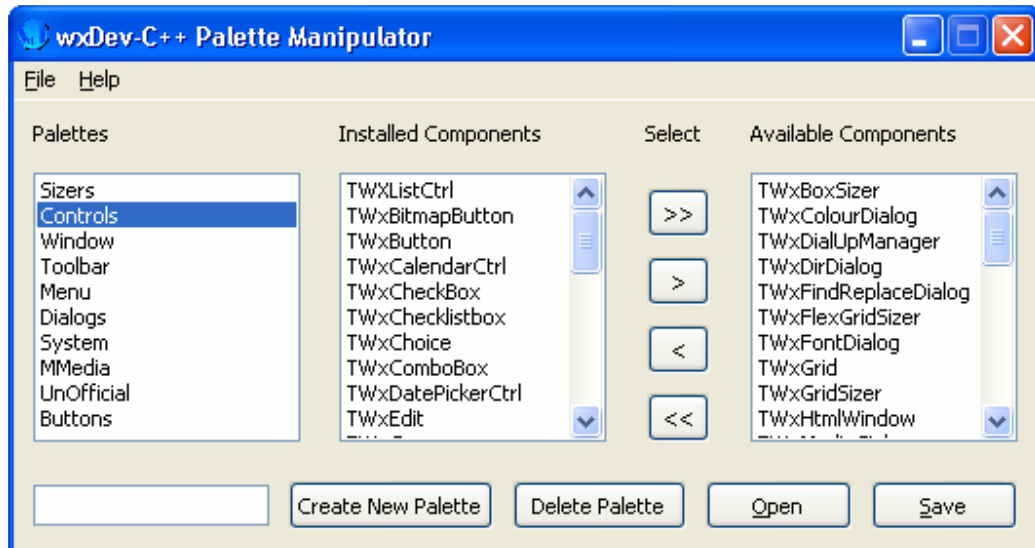






Figure 9.134 – Palette Manipulator

To move components to or from a palette use the ‘Select’ buttons.

-  Removes all components from the palette
-  Removes the selected component from the palette
-  Adds the selected component to the palette
-  Adds all components to the palette

To create a new palette enter the name in the bottom text box and click [Create New Palette].

The save button creates a backup of the previous palette file and save this one. The backup is confined to the last set of changes so if you have made an error restore from the backup before trying again.

Restart wxDev-C++ to see the changes.

Chapter 10 – Making It Work With Code

Introduction

This chapter is designed to help you make your beautiful GUI do something more than look nice. The only way to achieve this is by adding code. However it can be a problem finding where to add the necessary code. With wxDev-C++ you find that there are even certain places where your code will vanish. This chapter will help to guide you past such pitfalls.

Where to add your code (or where did it go)

We will start with an experiment to experience the disappearing code problem. We will follow this the reason why this happens. First we will start a new wxFrame project called VanishingCode.

Start a new wxFrame project called 'VanishingCode'.
Save it in a new folder called 'VanishingCode' within your project folder.
Accept the defaults for the frame and click [Create].
Turn to the tab that says 'VanishingFrm.h'.
Look for the line that says '////Header Include Start'.
After this line add the line '#include <wx/msgdlg.h>'.
Now turn to the tab that says 'VanishingFrm.cpp'.
Look for the line that says '////GUI Items Creation Start'.
Add a line after this with this code 'wxMessageBox(wxT("Hello
Everyone"), wxT("Doctor Nick says")) ;'.
Press <F9> to compile and run the application.

The following message box should pop up before the frame appears.



Figure 10.1 – The messagebox that appears at the start of the application

Now to demonstrate the disappearing code problem

Switch to the tab labelled 'VanishingCodeFrm.wxform'.
Select a button control from the palette and add to the form.
Press <F9> to compile and run the application.

This time the message box doesn't appear, all that appears is a frame containing a giant button. Look back in the two files you altered by hand, you will find that your code has vanished. So why has it vanished?

The answer is because we placed it between the two lines of code starting with '////'. The form designer uses these areas to add the code necessary to create the form, added the required header files, add events, etc. The IDE will replace all the code between these two lines each time it updates the file. So when we added the button the code was updated and our changes were removed.

So the moral is never add code between two lines starting with '////'. Generally there is a warning above these areas telling you not to add code there. However with all rules there is an exception, this occurs in the .cpp file within the event table block within this block is the lines '////Manual Code Start' and '////Manual Code End'. This is the one area where you should enter code between the two lines.

Responding to Events

The first thing we need to understand is the meaning of events. An event is something that happens. For example in real life the phone ringing or someone knocking at your door is an event. When an event happens you respond to it either you answer the phone or the door, or you hide and pretend no one is home. In GUI programming an event is more likely to be a button click, a window repainting, a timer going off and so on. Just like real life you decide what to do when an event occurs; you can even ignore it like the man knocking on your door.

So let's look at how we can respond to events. We will need a new project to demonstrate this.

- Start a new wxFrame project.
- Call it TheBigEvent and save it in a new folder with the same name.
- Accept the defaults for the frame.
- Drop an Edit control and a Button on the frame.
- Name the Edit control 'edtMessage' and the Button 'btnPushMe'.
- Change the label of the button to 'Push Me'.
- Select the button and in the Property Inspector select the Events tab.

There are two events listed OnClick and OnUpdateUI. The one we are going to look at is OnClick.

In the drop down box for the OnClick event choose <Add New Function>.

The IDE should take you to the newly generated event handler. Which will look like this.

```
/*  
 * btnPushMeClick
```



```

*/
void TheBigEventFrm::btnPushMeClick(wxCommandEvent& event)
{
    // insert your code here
}

```

From our knowledge of C++ programming we can see that the generated event handler is a member function of the frame. It takes a single parameter which is of the type `wxCommandEvent`. What we cannot see here is that the `wxCommandEvent` is one of several classes that derive from `wxEvent`. For many programs the parameter can be ignored since all we need to know about the event is that it happened, for other programs we need to know more information such as who sent the event. This information is held in the `wxEvent` derived argument.

We will not go much deeper into the mysteries of `wxWidgets` event handling than this. Instead we will look at responding to events. The IDE has very kindly provided us with a prompt on where to insert our code `// insert your code here`. We will accept this invitation.

Replace the line `// insert your code here` with
`edMessage->SetValue(wxT("The big event just happened"));`
 Press <F9> to compile and run the program.
 Try clicking on the button.

The result of running the program and pressing the button should look like this.

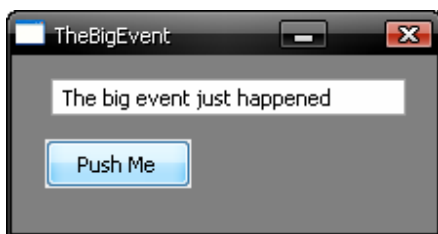


Figure 10.2 – The Big Event program after pressing the button

But how does the program know which function it should execute when the button is pressed? The answer lies near the top of the frames `.cpp` file in the event table. After you added the new `OnClick` function the table will look like this.

```

BEGIN_EVENT_TABLE(TheBigEventFrm,wxFrame)
    ///Manual Code Start
    ///Manual Code End

    EVT_CLOSE(TheBigEventFrm::OnClose)
    EVT_BUTTON(ID_BTNPUSHME,TheBigEventFrm::btnPushMeClick)
END_EVENT_TABLE()

```

There are two events listed here `EVT_CLOSE` and `EVT_BUTTON`. The first event occurs when you try to close the frame for example by clicking on the close button. As you can see

this calls the function `TheBigEventFrm::OnClose`. The second one which we just added has two parts the first part `ID_BTNPUSHME` is the ID Name of the component that is linked to the function which follows it `TheBigEventFrm::btnPushMeClick`. So when the button is clicked the application searches the event table for event linked to the buttons ID Name. When it finds an event of the correct type in this case `EVT_BUTTON` it calls the linked function.

This is a simple example of handling events, but in most cases it is all you will need to do. However `OnClick` is only one of many types of event. We will next look at a list of events and when they occur.

Types of Event

The `wxDev-C++` IDE names events after the action that causes them such as `OnClick`. However if you look through the `wxWidgets` documentation you wont find information about an event called `OnClick` since `wxWidgets` uses a different method to name events so in the documentation you will find `EVT_BUTTON`.



Figure 10.3 – The Button `OnClick` event in `wxDev-C++`

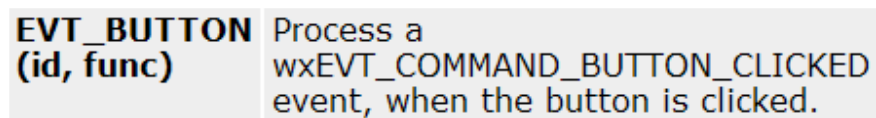


Figure 10.4 – The actual event in the `wxWidgets` documentation

Instead of listing every component and the events `wxDev-C++` allows you to alter I have created a list in the appendix that lists the names of the events, an example control that uses this event and when the event occurs. Following the description of when the event occurs is name of the event for you to look up in the `wxWidgets` help file.

Having taken a look at the event handling mechanism of `wxDev-C++` and `wxWidgets` let us put this into practice with our sample application. In the next section we are going to start adding code in order to get the whole thing up and running.

Sample Application Part 3 – The code

We now look at the most important part of the sample application. At present our program does nothing except start, there is no splash screen, no way to open dialogs and the buttons do nothing. In order to make it all work we need to write some code.

The Application Code

First we are going to start by altering the application code file to enable us to display a splash screen and to show start up tips. The file we need to edit is called HTMLEditApp.cpp. It present it looks like this:

HTMLEditApp.cpp

```
1  #include "HTMLEditApp.h"
2  #include "HTMLEditFrm.h"
3
4  IMPLEMENT_APP(HTMLEditFrmApp)
5
6  bool HTMLEditFrmApp::OnInit()
7  {
8      HTMLEditFrm* frame = new HTMLEditFrm(NULL);
9      SetTopWindow(frame);
10     frame->Show();
11     return true;
12 }
13
14 int HTMLEditFrmApp::OnExit()
15 {
16     return 0;
17 }
```

At the start of the file two files are included the header file for the declaration of the application class which derives from wxApp. This class is where the program starts. The other included file is the declaration of our main frame. This is needed so that the application class can create and display our frame.

When the application class starts it runs the function `OnInit()` this function then creates our frame and shows it. Since a splash screen is shown before the main screen we need to add the necessary code before line 8. We also need tell the application where to find our splash screen. So under line 2 add this line

```
#include "SplashFrm.h"
```

Then add these 3 lines before line 8.

```
SplashFrm* SplashFrame = new SplashFrm(NULL);
SetTopWindow(SplashFrame);
SplashFrame->DisplayFrame();
```

At present we can't compile the program since our splash frame doesn't have a function called `DisplayFrame()`, but we are going to implement this soon. Before we look at doing this we need add a couple more pieces of code.

Firstly before the code we just added add this line.

```
wxInitAllImageHandlers();
```

Since we will be working with images we need to initialise the image handlers this allows us to load and save different image types.

Finally we want to add the ability to show tips when the program starts. This has to be done after the frame has been created and shown. So before the line `return true;` in the function `OnInit()` add the following lines.

```
if (/*Normally check here to see if tips should shown*/1)
{
    wxTipProvider *tipProvider = wxCreateFileTipProvider("tips.txt", 0);
    wxShowTip(frame, tipProvider);
    delete tipProvider;
}
```

Within the `if` statement a proper application would check to see if the user wants to show tips or not. We put a 1 in here to make the check always return true. The application then looks for a text file called `tips.txt` in the same location as the application. Without this file the user will be shown an error message that the tips file cannot be found. Create a text file with the following contents.

tips.txt

```
Welcome to Simple HTML Editor.
This is a sample application written in wxWidgets and wxDev-C++.
It accompanies the book Programming with wxDev-C++.
```

Save this file in the same directory as the executable file.

In order to use the `wxTipProvider` you need to add the following include file to the list of include files.

```
#include <wx/tipdlg.h>
```

Next we are going to add the display function to the splash screen. After this we will be able to compile and run the application.

The Splash Screen

We need to edit two file to enable us to make the splash screen work as we wish. The first file is `SplashFrm.h`. I will not display all the contents of this file, but near the end you will find this function declaration `void CreateGUIControls();` . After this line add the following code.

SplashFrm.h

```
void SetWindowShape();
public:
    void DisplayFrame();
```

Next we need to provide the code to use these two functions, we will do this in the file `SplashFrm.cpp`.

The first line of code we need to add is in the function `void SplashFrm::CreateGUIControls()`. We need to add two lines after the auto generated code so look for the line

```
////GUI Items Creation End
```

This is around line 60. After this line add the following code.

```
SetSize(wxSize bmpSplashImage->GetBitmap().GetWidth(), bmpSplashImage->GetBitmap().GetHeight());
SetWindowShape();
```

The first line makes sure our frame is set to the same size as the bitmap we are going to display. The second calls our function `SetWindowShape()`. We are going to add the code for this function after the `OnClose` function around about line 69.

SetWindowShape()

```
void SplashFrm::SetWindowShape()
{
    //Convert image to a wxImage so that we can access pixel values
    wxImage TempImage = bmpSplashImage->GetBitmap().ConvertToImage();
    //Create region using top left pixel as transparent colour
    wxRegion region(bmpSplashImage->GetBitmap(),
wxColour(TempImage.GetRed(0,0),TempImage.GetGreen(0,0),TempImage.GetBlue(0,0)));
    //Set the window shape using new region
    SetShape(region);
}
```

I have added comments to explain what each line does. The whole function basically cuts out the splash screen frame to the same shape as the outline of the image we are using on the splash screen. We now need to add the code for the `DisplayFrame()` function. We will add this after the last function.

DisplayFrame()

```
void SplashFrm::DisplayFrame()
{
    int i = 0;
    Show(true);
    for(i = 0; i < 250; i+=5)
    {
        SetTransparent(i);
        Update();
        wxMilliSleep(15); //Fix by Fabian Wey
    }
    wxSleep(2);
    for(i = 255; i > 0; i-=5)
    {
        SetTransparent(i);
    }
}
```

```
Update();  
wxMilliSleep(15); //Fix by Fabian Wey  
}  
Destroy();  
}
```

This function first of all displays the frame. Then enters a loop which alters the frames transparency causing it to fade in. Then it pauses using `wxSleep()`, before using another loop to fade back out again. Finally it destroys itself.

Now we finally have a program that does something interesting. So let us have a look at what we have achieved.

Press F9 to compile and run.

The first thing you should see is the splash screen, this is shaped to the letters on the image. After this fades in and out the frame will display followed by the tip window. If you have created the `tips.txt` file in the correct place you should see this.



Figure 10.3 – The application displaying a tip window

Next we will look at coding the various dialogs before using them in our program.

The Dialogs

The first dialog we are going to look at is the hyperlink dialog. The reason for this is that it is the simplest. One of the first problems that many users come across is that wxDev-C++ creates the various components on a frame or dialog as private. This means that you cannot directly access members of the dialog from another frame such as your main frame. This means we are going to have to write accessor functions. We will start in `InsertHyperlinkDlg.h`.

We need a get/set method for the combobox on the dialog, so that we can alter and retrieve its contents from our main frame. Look for the line `void CreateGUIControls()`; this should be around line 77. After this line add the following code.

InsertHyperlinkDlg.h

```
public:
    const wxString GetHyperlink(){return cmbHyperLink->GetValue();};
    void SetHyperlinks(const wxArrayString AnchorArray){cmbHyperLink->Append(AnchorArray);};
```

We create these as public so that we can access them the first function returns the currently selected item in the hyperlink combobox. The second function takes an array of wxStrings to set the choices allowed by the combobox.

We will now look at coding the insert image dialog. We again need to add code to the header file after the line `void CreateGUIControls()`; this is around line 98.

InsertImageDlg.h

```
public:
    const int GetImageWidth(){return spnWidth->GetValue();};
    const int GetImageHeight(){return spnHeight->GetValue();};
    const int GetBorderWidth(){return spnBorder->GetValue();};
    const wxString edtImageSource(){return txtImageSource->GetValue();};
    const wxString edtAlternativeText(){return txtAlternativeText->GetValue();};
```

We only want to retrieve values from this dialog so all the functions simply return values from the various controls. However we also need add code for the operation of the button `btnBrowseForImage`. To do this

Change tabs to the `InsertImageDlg.wxform` tab.

Select the `btnBrowseForImage`

In the Property Inspector change to the Events tab.

In the OnClick combobox choose <Add New Function> from the choices.

You should automatically be switched the the new skeleton function.

After the line `// insert your code here` add the following code.

btnBrowseForImageClick()

```
if(WxOpenFileDialog1->ShowModal())
{
    wxFileName TempFilename(dlgFileChooser->GetPath());
    //Extract the extension part of the file name
    wxString TempExtensionString(TempFilename.GetExt());
    wxBitmap TempBitmap;
    //Use the extracted extension to decide what file type to load
    if(TempExtensionString.CmpNoCase(wxT("PNG")) == 0)
    {
        TempBitmap.LoadFile(dlgFileChooser->GetPath(),wxBITMAP_TYPE_PNG);
    }
    else if((TempExtensionString.CmpNoCase(wxT("JPG")) == 0) ||
(TempExtensionString.CmpNoCase(wxT("JPEG")) == 0))
    {
        TempBitmap.LoadFile(dlgFileChooser->GetPath(),wxBITMAP_TYPE_JPEG);
    }
    else if(TempExtensionString.CmpNoCase(wxT("GIF")) == 0)
    {
        TempBitmap.LoadFile(dlgFileChooser->GetPath(),wxBITMAP_TYPE_GIF);
    }
    //Fill the controls with relevant values
    spnHeight->SetValue(TempBitmap.GetHeight());
    spnWidth->SetValue(TempBitmap.GetWidth());
    edtImageSource->SetValue(dlgFileChooser->GetPath());
    //Display a preview of the image chosen
    imgPreview->SetBitmap(TempBitmap);
}
```

We also need to add a header file to allow us to use the wxFileName class so after the line `////Header Include End` (about line 16) add this line.

```
#include <wx/filename.h>
```

In this function we use the class wxFileName which allows us to divide a filename into different parts, we use it to easily retrieve the file extension. Next we use the extension part to determine what type of image to load, this is necessary for the second part of the wxBitmap LoadFile() function. Once the image is loaded we fill the controls with values such as image size and location. Finally we show a preview of the image.

Our final dialog is the create table dialog. This is the most complex of the dialogs and we will start with the CreateTableDlg.h file. Once again we need to write accessor function as well as a couple of button click event handlers. As before add the following code after the CreateGUIControls() function which is around line 118.

CreateTableDlg.h

```
public:
    const int GetRows(){return spnRows->GetValue();};
    const int GetCols(){return spnCols->GetValue();};
    const int GetBorderSize(){return spnBorder->GetValue();};
    const int GetCellSpacing(){return spnCellSpacing->GetValue();};
    const int GetCellPadding(){return spnCellPadding->GetValue();};
    const int GetTableWidth(){return spnTableWidth->GetValue();};
```



```

    const wxString GetTableAlignment(){return rdbTableAlignment-
>GetStringSelection();};
    const bool IsColourSelected(){return rdbTableBackground->GetSelection() ==
0?true:false;};
    const bool IsImageSelected(){return rdbTableBackground->GetSelection() ==
1?true:false;};
    const wxString GetColour(){return pnlColour-
>GetBackgroundColour().GetAsString(wxC2S_HTML_SYNTAX);};
    const wxString GetImage(){return edtImageSource->GetValue();};

```

Of some interest here are the IsColourSelected and IsImageSelected functions which use the shorthand if statement to return a true or false Boolean value. The GetColour function could also use some explaining, it gets the background colour of the panel which is returned as a wxColour, we then call the wxColour's function GetAsString which we can ask to return the string in HTML syntax which is what we need for web pages.

Now we need to turn to the CreateTableDlg.wxform tab. Select the button btnChooseColour and in the events section create a new OnClick event. Add the following code after the `// insert your code here` line.

btnChooseColourClick

```

if(dlgColorChooser->ShowModal() == wxID_OK)
{
    //Set the panel colour to match the value held by the colour dialog
    pnlColor->SetBackgroundColour(dlgColorChooser->GetColourData().GetColour());
    //Refresh the panel to show the new colour
    pnlColor->Refresh();
    //Set radio selection to use colour
    rdbTableBackground->SetSelection(0);
}

```

We will repeat the above procedure for the btnChooseImage button. In the newly created OnClick function add the following code.

btnChooseImageClick

```

if(dlgFileChooser->ShowModal() == wxID_OK)
{
    edtImageSource->SetValue(dlgFileChooser->GetPath());
    //Set radio selection to use image
    rdbTableBackground->SetSelection(1);
}

```

Time to relax and have a drink or your choice. After all that coding we have several dialogs that do nothing. Time to connect them to our main frame. Before continuing press F9 to check that everything compiles OK. If there are any errors correct them before continuing.

The Main Frame

Okay get your fingers ready for a real workout in this section. We will mostly be creating OnClick event function for buttons and menu items, but we will also be coding some helper functions. After all if a button and menu does the same thing why code it twice why not just write one function and get the button and the menu to both call it.

We will start with the helper functions, in the HTMLEditFrm.h after the CreateGUIControls function (about line 164) add the following function declarations.

HTMLEditFrm.h

```
public:
    void UpdateHTML();
    void OpenFile();
    void SaveFile();
    void NewFile();
    bool DoFileSaveCheck();
    void WrapTextInTag(const wxString& TagType);
    void InsertText(const wxString& Text);
```

We also need to add a wxString variable to hold the name of our currently open file, this will be used to add the name to the list of most recently used (MRU) files. Around line 98 after the line `////GUI Control Declaration End` add this line.

```
wxString MRUFile;
```

We now need to add the function definitions. These go in the HTMLEditFrm.cpp change to this tab and scroll to the bottom of this file then add the following code.

HTMLEditFrm.cpp

```
void HTMLEditFrm::UpdateHTML()
{
    //Update and refresh the text on the HTML preview windows
    htmCombinedPreview->SetPage(mmoTextEditor->GetValue());
    htmPreview->SetPage(mmoTextEditor->GetValue());
    //Something has changed so mark the text as altered in the memo
    mmoTextEditor->MarkDirty();
}

void HTMLEditFrm::OpenFile()
{
    // Check if we need to save current file before opening new
    if(DoFileSaveCheck())
    {
        //Show open file dialog
        if(dlgFileOpen->ShowModal() != wxID_CANCEL)
        {
            //If we have a file name add it to the most recently used menu
            if(!MRUFile.IsEmpty())
            {
                m_fileHistory->AddFileToHistory(MRUFile);
            }
            //Load in the file
            mmoTextEditor->LoadFile(dlgFileOpen->GetPath());
            //Save the file path in the most recently used files
```

```

        MRUFile = dlgFileOpen->GetPath();
        //Update HTML preview
        UpdateHTML();
        //And reset text editor
        mmoTextEditor->DiscardEdits();
    }
}

void HTMLEditFrm::SaveFile()
{
    //Set default filename and show save dialog
    dlgFileSave->SetFilename(MRUFile);
    if(dlgFileSave->ShowModal() != wxID_CANCEL)
    {
        //Save file and add filename to most recently used files
        mmoTextEditor->SaveFile(dlgFileSave->GetPath());
        MRUFile = dlgFileSave->GetPath();
        m_fileHistory->AddFileToHistory(dlgFileSave->GetPath());
    }
}

void HTMLEditFrm::NewFile()
{
    //Check if we need to save the file
    if(DoFileSaveCheck())
    {
        //Clear reset text and HTML displays
        mmoTextEditor->Clear();
        UpdateHTML();
        mmoTextEditor->DiscardEdits();
    }
}

bool HTMLEditFrm::DoFileSaveCheck()
{
    bool RetVal = true;
    //If we have unsaved changes in the text editor
    if(mmoTextEditor->IsModified())
    {
        //Check if we should save the changes
        if(dlgSaveChanges->ShowModal() == wxID_YES)
        {
            //Set a default filename and save file
            dlgFileSave->SetFilename(MRUFile);
            if(dlgFileSave->ShowModal() != wxID_CANCEL)
            {
                mmoTextEditor->SaveFile(dlgFileSave->GetPath());
                MRUFile = dlgFileSave->GetPath();
                //Add to list of most recently used files
                m_fileHistory->AddFileToHistory(MRUFile);
            }
            else
                RetVal = false;
        }
    }
    else
    {
        if(!MRUFile.IsEmpty())
        {
            //Add to list of most recently used files
            m_fileHistory->AddFileToHistory(MRUFile);
        }
    }
}

```

```

    }
}

return RetVal;
}

void HTMLEditFrm::WrapTextInTag(const wxString& TagType)
{
    //Create variables to hold position of currently selected text
    long PosFrom = 0, PosTo = 0;
    //Get position of currently selected text
    mmoTextEditor->GetSelection(&PosFrom,&PosTo);
    wxString TempString;
    //Add tag start to temp string
    TempString << wxT("<") << TagType << wxT(">");
    //Add currently selected text to temp string
    TempString << mmoTextEditor->GetStringSelection();
    //Add tag end to temp string
    TempString << wxT("</") << TagType << wxT(">");
    //Replace currently selected text with contents of temp string
    mmoTextEditor->Replace(PosFrom,PosTo,TempString);
    //And update controls
    UpdateHTML();
}

void HTMLEditFrm::InsertText(const wxString& Text)
{
    //Create variables to hold position of cursor
    long PosFrom = 0, PosTo = 0;
    //Get current selection
    mmoTextEditor->GetSelection(&PosFrom,&PosTo);
    //And replace with the text to insert
    mmoTextEditor->Replace(PosFrom,PosTo,Text);
    //Then update controls
    UpdateHTML();
}
}

```

The idea of helper function like these are not only because I am lazy and don't like to type the same code several times, but also because it makes it easier to change code later, If I want to change the way I load a file I don't want to hunt for the load file button press event code, then the load file menu event code and try to keep them the same. I want to look for and alter one function only.

At this point we want to be sure that everything is still OK. So press F9 to compile and run the program. If there are any errors now is the time to fix them before we move on to the more exciting task of connecting up the controls to code.

Making it all work

We will start with showing the about dialog. We want to do this when the Help->About menu item is selected. To do this

- Change to the HTMLEditFrm.wxform tab.
- Select the mnuMainBar control
- On the Properties tab select Edit MenuItems
- Expand the &Help option and select the &About option

Click the [Edit] button.
Next the OnMenu combobox click the [Create] button.
Change the function name to MnuAboutClick.
Click [OK].
Click [Apply] then [OK].
Then change to HTMLEditFrm.cpp and scroll to the bottom to our new function.
Add the following code after the `// insert your code here` line.

MnuAboutClick

```
AboutBoxDlg TempAboutBoxDlg(this);  
TempAboutBoxDlg.ShowModal();
```

Now follow the same procedure for the other menu items

MnuNewClick

```
NewFile();
```

MnuOpenClick

```
OpenFile();
```

MnuSaveClick

```
SaveFile();
```

MnuPageSetupClick

```
dlgPrint->PageSetup();
```

MnuPrintPreviewClick

```
dlgPrint->PreviewText(mmoTextEditor->GetValue());
```

MnuPrintClick

```
dlgPrint->PrintText(mmoTextEditor->GetValue());
```

MnuExitClick

```
Close();
```

MnuUndoClick

```
mmoTextEditor->Undo();  
UpdateHTML();
```

MnuRedoClick

```
mmoTextEditor->Redo();  
UpdateHTML();
```

MnuCutClick

```
mmoTextEditor->Cut();  
UpdateHTML();
```

MnuCopyClick

```
mmoTextEditor->Copy();
```

MnuPasteClick

```
mmoTextEditor->Paste();  
UpdateHTML();
```

That is it for the menu code, as you can see the controls provided by wxWidgets do most of the hard work for us. We only need to call the functions they provide such as cut, paste, etc.

Since we are now starting to use the dialogs we created earlier we need to tell the compiler where to find the code. Near the top of the HTMLEditFrm.cpp file under the line `////Header Include End` which is around line 46 add the following list of include files.

```
#include "InsertHyperlinkDlg.h"  
#include "InsertImageDlg.h"  
#include "CreateTableDlg.h"  
#include "AboutBoxDlg.h"
```

Now to check if your hard work has been successful. Press <F9> to compile and run the application. If there any compilation errors now is the time to fix them. Try using the different menus and see the results.

Next we are going to look at the main toolbar. For the first seven buttons I am going to leave you to create the OnClick functions and write the code (Hint the code is the same as for the menu items which have the same purpose). We are now going to look at the cmbHeadings combobox. The event we need to write the code for is the OnUpdated event. This occurs when the user makes a new selection. The code is as follows

cmbHeadingsUpdated

```
//Create a temporary string and send it the H tag plus a number derived from  
//the users selection, add 1 since the combo contents are numbered from zero  
WrapTextInTag(wxString() << wxT("H") << (cmbHeadings->GetCurrentSelection() +  
1));
```

Thanks to our helper function `WrapTextInTag` we can achieve a lot in one line of code. We are going to make use of the `InsertText` helper function for the alignment buttons. The `OnClick` function for each of these is as follows.

btnAlignLeftClick

```
InsertText(wxT(" ALIGN=\"left\""));
```

btnAlignCenterClick

```
InsertText(wxT(" ALIGN=\"center\""));
```

btnAlignRightClick

```
InsertText(wxT(" ALIGN=\"right\""));
```

Once again the `InsertText` function reduces the amount of code we need. The backslashes are needed to escape the quote signs since we need to display them as part of the alignment property.

We are now finished with the top toolbar, as before press <F9> to compile and run the application and check that there are no coding errors. Once you are satisfied we are going to move on to the side toolbar.

The first three buttons are fairly simple and work fairly like the header combobox. We will create them next. The code for the three of them follows.

btnEmboldenTextClick

```
WrapTextInTag(wxT("B"));
```

btnItaliciseTextClick

```
WrapTextInTag(wxT("I"));
```

btnUnderlineTextClick

I will leave this one for you it is the same as the other two the only different is that the tag letter is “U”. The next button is the font button, this is much more complex since we need to retrieve to values from the font dialog. Create a new `OnClick` function and add the following code.

btnChangeFontClick

```
if(dlgFont->ShowModal())  
{  
    wxFontData FontData = dlgFont->GetFontData();
```

```

wxFont Font = FontData.GetChosenFont();
long PosFrom = 0, PosTo = 0;
mmoTextEditor->GetSelection(&PosFrom,&PosTo);
wxString TempString;
TempString = wxT("<FONT COLOR=\");
TempString << FontData.GetColour().GetAsString(wxC2S_HTML_SYNTAX);
TempString << wxT("\") << wxT(" FACE=\");
TempString << Font.GetFaceName() << wxT("\");
TempString << wxT(" POINT-SIZE=\");
TempString << Font.GetPointSize() << wxT("\>");
TempString += mmoTextEditor->GetStringSelection();
TempString += wxT("</FONT>");
mmoTextEditor->Replace(PosFrom,PosTo,TempString);
UpdateHTML();
}

```

The next button creates a horizontal line the code follows.

btnInsertHorizontalLineClick

```
InsertText(wxT("<HR>"));
```

The next button creates anchor points. The user is prompted for the name of the anchor point, this is used to build the anchor point. The anchor point is added to a list of anchor points for use in the insert hypertext dialog. The code for this function follows.

btnCreateAnchorPointClick

```

//Create variables to hold position of selected text
long PosFrom = 0, PosTo = 0;
//Get the text selection
mmoTextEditor->GetSelection(&PosFrom,&PosTo);
wxString TempString , TempAnchorString;
//Get name of anchor point from the user
TempAnchorString << wxGetTextFromUser(wxT("Enter a name for your anchor
point"),wxT("Anchor point creator"));
//Create temporary string to build anchor point
TempString << wxT("<A NAME=\") << TempAnchorString << wxT("\>");
TempString << mmoTextEditor->GetStringSelection() << wxT("</A>");
//Write the anchor point
mmoTextEditor->Replace(PosFrom,PosTo,TempString);
//Update the HTML controls
UpdateHTML();
//Create an anchor point and add to list of anchor points
TempAnchorString.Prepend(wxT("#"));
AnchorArray.Add(TempAnchorString);

```

This function requires that we add another variable AnchorArray to the variables held by the frame. Under the wxString variable MRUFile in the HTMLEditFrm.h file add the line.

```
wxArrayString AnchorArray;
```


Tied in with the create anchor button is the create hyperlink button which we shall look at next. This uses the array of anchor points to populate the combobox on the InsertHyperlinkDialog. The code for this button follows.

btnInsertHyperlink

```
//Create a new InsertHyperlinkDialog and populate with anchor points
InsertHyperlinkDlg TempInsertHyperlinkDlg(this);
TempInsertHyperlinkDlg.SetHyperlinks(AnchorArray);
//Show the dialog
if(TempInsertHyperlinkDlg.ShowModal() == wxID_OK)
{
    long PosFrom = 0, PosTo = 0;
    mmoTextEditor->GetSelection(&PosFrom,&PosTo);
    //Create the hyperlink string
    wxString TempString;
    TempString << wxT("<A HREF=\"");
    TempString << TempInsertHyperlinkDlg.GetHyperlink();
    TempString << wxT(">");
    TempString << mmoTextEditor->GetStringSelection();
    TempString << wxT("</A>");
    //Insert the hyperlink string
    mmoTextEditor->Replace(PosFrom,PosTo,TempString);
    //And update the HTML controls
    UpdateHTML();
}
```

The next button is the insert table button, once again this will use one of our custom dialogs.

btnInsertTable

```
CreateTableDlg TempCreateTableDialog(this);
if(TempCreateTableDialog.ShowModal() == wxID_OK)
{
    long PosFrom = 0, PosTo = 0;
    mmoTextEditor->GetSelection(&PosFrom,&PosTo);
    wxString TempString;
    //Start to create the table string
    TempString << wxT("<TABLE ALIGN=\"");
    TempString << TempCreateTableDialog.GetTableAlignment();
    //Check if user wishes to use a background image or colour
    if(TempCreateTableDialog.IsImageSelected())
    {
        TempString << wxT("> BACKGROUND=\"");
        TempString << TempCreateTableDialog.GetImage();
    }
    else if(TempCreateTableDialog.IsColourSelected())
    {
        TempString << wxT("> BGCOLOR=\"");
        TempString << TempCreateTableDialog.GetColour();
    }
    //Start to add the table attributes
    TempString << wxT("> BORDER=\"");
    TempString << TempCreateTableDialog.GetBorderSize();
    TempString << wxT(" CELLSPACING=\"");
    TempString << TempCreateTableDialog.GetCellPadding();
    TempString << wxT(" CELLPADDING=\"");
    TempString << TempCreateTableDialog.GetCellSpacing();
}
```

```

TempString << wxT( "\ " WIDTH=\ " );
TempString << TempCreateTableDialog.GetCellSpacing();
TempString << wxT( "\ ">\n" );
//Loop through to create all the table rows
for(int i = TempCreateTableDialog.GetRows();i>=0;i--)
{
    TempString << wxT( "<TR>\n" );
    //Loop through to create all the table columns
    for(int j = TempCreateTableDialog.GetCols();j>=0;j--)
    {
        TempString << wxT( "<TD>\n" );
        TempString << wxT( "</TD>\n" );
    }
    TempString << wxT( "</TR>\n" );
}
//End the table
TempString += wxT( "</TABLE>\n" );
//Insert the table string
mmoTextEditor->Replace( PosFrom, PosTo, TempString );
//Update the HTML controls
UpdateHTML();
}

```

We now only have one button left to write code for the insert image button. Follow the usual procedure and add the following code to the OnClick function.

btnInsertImageClick

```

//Show insert image dialog
InsertImageDlg TempInsertImageDialog( this );
if( TempInsertImageDialog.ShowModal() == wxID_OK )
{
    //Create string to hold image details
    wxString TempString = wxT( "<IMG" );
    TempString << wxT( " BORDER=\ " );
    TempString << TempInsertImageDialog.GetBorderWidth();
    TempString << wxT( "\ " HEIGHT=\ " );
    TempString << TempInsertImageDialog.GetImageHeight();
    TempString << wxT( "\ " WIDTH=\ " );
    TempString << TempInsertImageDialog.GetImageWidth();
    TempString << wxT( "\ " SRC=\ " );
    TempString << TempInsertImageDialog.GetImageSource();
    TempString << wxT( "\ " TITLE=\ " );
    TempString << TempInsertImageDialog.GetAlternativeText();
    TempString << wxT( "\ "> );
    //Add to the text editor
    mmoTextEditor->WriteText( TempString );
    //Update the HTML controls
    UpdateHTML();
}

```

Finally we need to make a minor change to the AboutBoxDlg.cpp file. At present if we click on the hyperlinks on the about box we are taken to the wxDev-C++ website. This is not what we want so look for the line `////GUI Items Creation End` (around line 80) and add the two following lines of code.

```

hypProjectSite->SetURL( wxT( "http://wxdevcpp-book.sourceforge.net" ) );
hypIconsFrom->SetURL( wxT( "http://tango.freedesktop.org/Tango_Desktop_Project" ) );

```

And that is it for the controls. However at the moment the application still has a few major flaws. Firstly the HTML does not update as the user types, secondly when the user presses enter the HTML tag
 should be added to the text editor, thirdly there are no prompts to save the current file when the user exits the application and finally the most recently used file menu entry does nothing. We will fix this next.

Tidying up

The first problem is easy to solve we already have a function UpdateHTML() we just need to call it in the correct place. The function we need is called OnUpdated and is part of the mmoTextEditor text control. Create a new function and add the line

mmoTextEditorUpdated

```
UpdateHTML();
```

This has now created a new problem, try compiling and running the application and you may find that it crashes after the splash screen shows. The reason for this is that when the text control is created it sends an update event, our UpdateHTML code then tries to update the HTML controls which are not yet created.

We now need to modify the UpdateHTML function. Modify it to look like the following.

UpdateHTML()

```
//Check the GUI controls have been created before continuing
if(SetupComplete)
{
    //Update and refresh the text on the HTML preview windows
    htmCombinedPreview->SetPage(mmoTextEditor->GetValue());
    htmPreview->SetPage(mmoTextEditor->GetValue());
    //Something has changed so mark the text as altered in the memo
    mmoTextEditor->MarkDirty();
}
```

We now need to create the variable SetupComplete and alter it in the correct place. We will create it in the HTMLEditfrm.h file. This will go after the line `////GUI Control Declaration End` and looks like this.

```
bool SetupComplete;
```

We now need to make an alteration in the function HTMLEditFrm::CreateGUIControls() that the IDE has created for us in the file HTMLEditFrm.cpp. Alter the function to look like the following.

```
SetupComplete = false;
CreateGUIControls();
SetupComplete = true;
```

Now we are ready to continue. This is a good example of hard to find bugs that arise from seemingly safe code. After all we only called a function that we have called many times before. This is where a debugger is extremely helpful as the backtrace can show you exactly where the program crashed.

The second problem requires us to create another event handler for the same control. This time we need to modify the OnEnter event. Create a new event and add the following code

mmoTextEditorEnter

```
mmoTextEditor->WriteText(wxT("<BR>"));
```

Next we need to prompt the user to save the file when they close the program if the file has changed since they last saved it. To do this we need to modify the OnClose function that the IDE created for us automatically. At the moment it only contains one line of code Destroy(); We need to alter it to look like this.

```
//Save the most recently used files
m_fileConfig->SetPath(wxT("/RecentFiles"));
m_fileHistory->Save(*m_fileConfig);
m_fileConfig->SetPath(wxT("../"));

//Check if we can veto this event, we can't if operating system says no
if(event.CanVeto())
{
    //Check if we have any unsaved code
    if(mmoTextEditor->IsModified())
    {
        //Check if the user actually wants to save changes
        if(dlgSaveChanges->ShowModal() == wxID_YES)
        {
            //Set default filename
            dlgFileSave->SetFilename(MRUFile);
            //Show the save file dialog
            if(dlgFileSave->ShowModal() != wxID_CANCEL)
            {
                mmoTextEditor->SaveFile(dlgFileSave->GetPath());
                Destroy();
            }
        }
        else
            Destroy();
    }
    else
        Destroy();
}
else
    Destroy();
```

The final problem is a bit more tricky we need to create our own event handler and add it manually. The function declaration needs to be added in HTML>EditFrm.h just before the end of class }; line which is around line 212. Add the following line

```
void OnMRUFile(wxCommandEvent & event);
```

We then need to write the function definition we will add this at the end of the file HTML>EditFrm.cpp. Add the following function.

```
void HTML>EditFrm::OnMRUFile(wxCommandEvent & event)
{
    //See if we need to save the current contents of the text editor
    if(DoFileSaveCheck())
    {
        //Get the file name using the event ID value
        MRUFile = m_fileHistory->GetHistoryFile(event.GetId() - wxID_FILE1);
        //Load the file held in the variable MRUFile
        mmoTextEditor->LoadFile(MRUFile);
        //Update the HTML controls
        UpdateHTML();
        //Mark the text editor as unchanged
        mmoTextEditor->DiscardEdits();
    }
}
```

Finally we need to connect this event handler to the event mechanism. We want to catch a range of events IDs since each MRU will send a different ID. We will add the code near the top of the HTML>EditFrm.cpp file after the line `////Manual Code Start` which is around line 59. Add the following line

```
EVT_MENU_RANGE(wxID_FILE1, wxID_FILE9, HTML/EditFrm::OnMRUFile)
```

This connects our event handler to a range of menu event IDs numbering from `wxID_FILE1` to `wxID_FILE9`.

That is it we are done try pressing F9 to try out the completed editor.

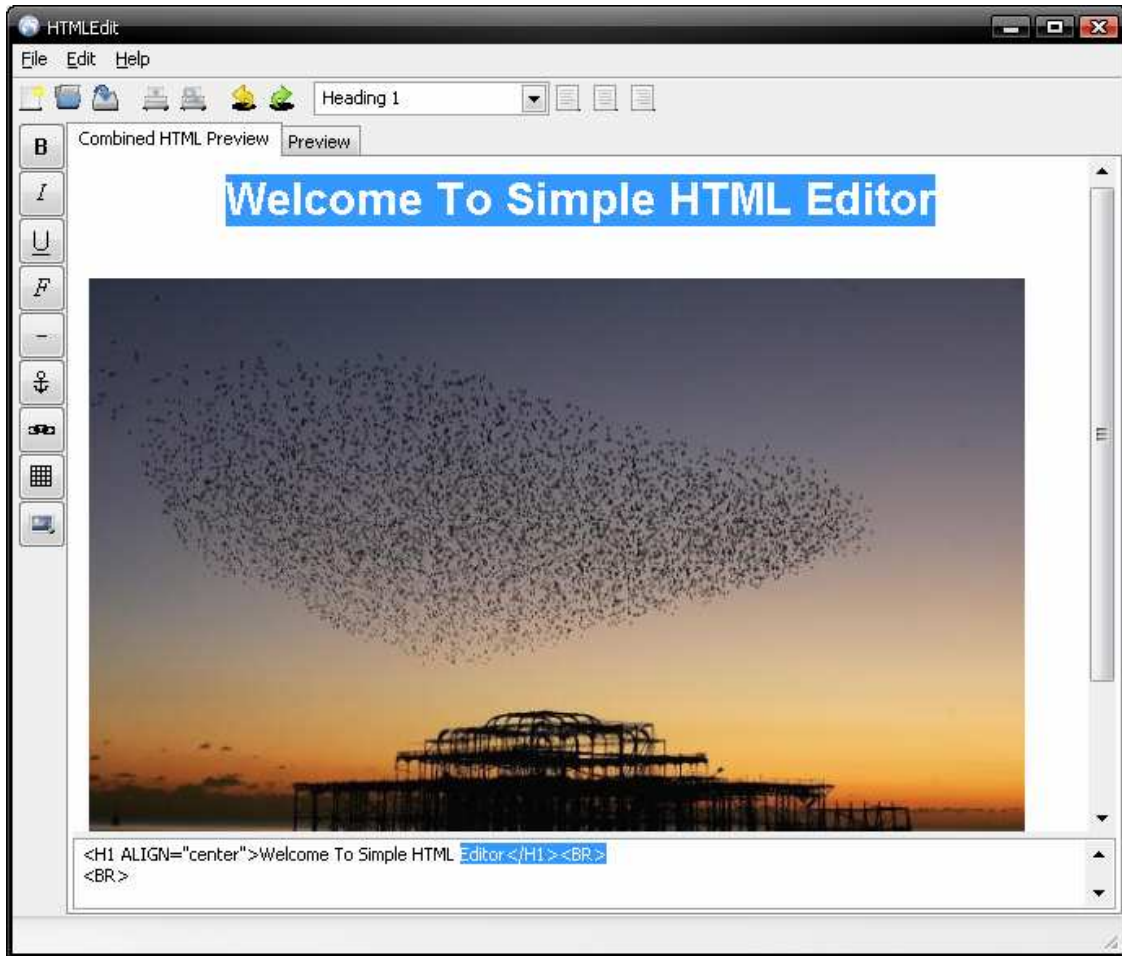


Figure 10.3 – The editor in action

This chapter has shown us how to add code to our GUI's in order to make them work. The following chapter is one we have referred to already and deals with making our application look and behave in a professional manner. Many of the points covered by this chapter have already been incorporated in our application. However it contains a few pointers to other improvements that could be incorporated.

Chapter 11 – Guidelines for Professional Looking Programs

Introduction

This chapter has very little to do with wxDev-C++ and more to do with programming in general. It is one thing to write a program, it is another to make it act and look professional. This applies the source code as much as to the look of the GUI. The GUI is important for your end users, the source code is important for you and other maintainers of your code.

This chapter will touch on many of the areas that can improve your programs and make your source code much nicer to work with and maintain.

Naming Conventions

There are almost as many different naming conventions as there are programmers and debates about the best convention can become quite heated. Basically a naming convention means to give your variables meaningful names that help improve the readability of your code. The convention you choose to use may be dictated by your project specification, or it may be one you have developed by yourself. There is no set right way, but there are definite wrong ways.

- 1. Choose a convention and stick to it throughout your coding.**

For example you may choose to start all integer variables with `int` and floats with `flt`.

Right	Wrong
<code>int intNumberOfVegetables;</code>	<code>int intVeg;</code>
<code>int intNumberOfCustomers;</code>	<code>int C;</code>
<code>float fltTotalCost;</code>	<code>float ftot;</code>
<code>float fltTempCost;</code>	<code>float cst;</code>

As you can see the first column is easier to understand you can see at a glance what each variable type is wherever it appears this makes it harder for example to accidentally assign a float to an int.

The second column shows code that would be more difficult to understand. For one thing some variable show their type and other don't. It is also harder to understand because of the following rule.

- 2. Give each variable a meaningful name.**

In the table above the first column shows variable that have been given names that explain their purpose. It is true that it takes longer to write 'NumberOfCustomers' than just 'C', but it a years time when you look back at your code it will be a lot easier to remember what that variable is intended to do.

A common exception to this rule is the variables used for loops. It is common to call the variable 'i' if you have a second loop within the first the variable name 'j' is often used and so on. Also variables used for coordinates are often called just x or y.

3. Make variable names easier to read.

If a name contains multiple words it is common to break them up in some way to make them more readable. This is either done by using capital letters for each word or separating them with underscores.

Right	Wrong
<pre>int intNumberOfVegetables; int int_number_of_vegtables; int IntNumberOfVegtables;</pre>	<pre>int intnumberofvegetables;</pre>

4. Use nouns for class names.

Since a class is a representation of an object it nearly always makes sense to use a noun to describe it.

Right	Wrong
<pre>class Car; class EditBox;</pre>	<pre>class IsItADrivingThing; class HasItWheels;</pre>

5. Use verbs for method or function names.

Since functions carry out actions it nearly always makes sense to use a verb to describe them. If a function returns a Boolean value verbs like is, has, can, etc are most suitable.

Right	Wrong
<pre>string MakeNameString(); int CountToTen(); bool IsNameValid(); bool CanQuit();</pre>	<pre>string NameString(); int Ten(); bool NameValid(); bool Quit();</pre>

6. Make constant variables more obvious.

Since constants name a value that will not change it makes sense to name them in such a way that the code looks wrong if you try to use a constant as a

normal variable and assign a value to it. This is usually achieved by using all capital letters for the constant name.

Right	Wrong
<pre>int DAYS_IN_THE_YEAR = 365; float PI = 3.141592;</pre>	<pre>int days_in_the_year = 365; float pi = 3.141592;</pre>

The second point can be extended for GUI components, by adding an abbreviated form of the component name to the start of the variable. For example

Component	Suggested Name
<code>wxTextCtrl</code>	<code>wxTextCtrl txtSomeName;</code>
<code>wxPanel</code>	<code>wxPanel pnlSomeName;</code>
<code>wxButton</code>	<code>wxButton btnSomeName;</code>
<code>wxStaticLabel</code>	<code>wxStaticLabel lblSomeName;</code>
<code>wxCheckBox</code>	<code>wxCheckBox chkSomeName;</code>
<code>wxRadioButton</code>	<code>wxRadioButton rdbSomeName;</code>

This covers the basics but there is a lot more to naming conventions. To find out more simply type 'Naming Conventions' into any search engine. Some links worth looking at are:

http://en.wikipedia.org/wiki/Identifier_naming_convention

<http://www.joelonsoftware.com/articles/Wrong.html>

http://en.wikipedia.org/wiki/Hungarian_notation

Remember the most important thing is to settle on or develop a convention you are comfortable with and be *consistent* with it.

Mnemonics and Keyboard Accelerators

Mnemonics

Mnemonics are letters in a menu or on certain controls that are underlined. They allow the user to hold down the ALT key and press the underlined letter on the keyboard to simulate clicking on that menu entry or control. This works on unix and Windows platforms, at present this is not implemented on Macintosh platforms.

We saw an example of using mnemonics when building our first wxWidgets program. We used them while building the menu. Do you remember those using '&' in the captions. The letter that the '&' proceeds is used as the mnemonic. Often this will be the first letter of the menu entry, but as mnemonics need to be unique sometimes the first letter is unavailable in which case other letters with a strong association, e.g. 'E&xit'. Try to avoid letters that descend as the underline can be hidden by these. E.g. 'g', 'p', 'j'. Avoid narrow character such as 'i'.

Some of these mnemonics are conventions, for example 'E&xit', '&File', 'Cu&t' and so on. If you are unsure what to use, try looking at other well known programs and see what they use. Other than that some general rules to help you are. Try to use the first or second letter in a word. E.g. '&File', '&Help'. Try to use a distinctive constant. E.g. 'E&xit'.

Generally Mnemonics are used on all controls that can contain text captions and can be clicked upon. Common exceptions are OK and Cancel buttons which should be mapped to ENTER and DELETE.

There is another convention that applies to menu items that is worth mentioning here. If clicking on the menu item will open a dialog box then the convention is to end the menu items caption with '...' for example 'Properties ...'

Keyboard Accelerators

Keyboard Accelerators are keys or key combinations that the user can press to activate certain functions. We have already come across these in wxDev-C++ itself. For example when we press <F9> to compile and run, F9 is a keyboard accelerator. Again we used them in our first wxWidgets program. This was the part after the menu name that began with '\t' everything after the '\t' was the keyboard accelerator.

Once again keyboard accelerators follow conventions, a table with a few of these follows. Once again if in doubt check other professional programs and see what they have used.

New	Ctrl+N	Redo	Shift + Ctrl + Z	Find	Ctrl+F
Open	Ctrl+O	Cut	Ctrl+X	Select All	Ctrl+A
Save	Ctrl+S	Copy	Ctrl+C	Search Again	F3
Print	Ctrl+P	Paste	Ctrl+V	Goto Line	Ctrl+G
Undo	Ctrl + Z	Delete	Del	Help	F1

You may think that all this mnemonic and keyboard accelerator nonsense is just useless bells and whistles for a program. If so just wait until the day the mouse attached to your computer stops working. That is the day you will thank someone for spending the time to make sure that you can operate your computer using only the keyboard.

Working with Multiple Source Code Files

As you program you will notice that some programs come in one massive source code file and others are split into multiple smaller files, some with .c or .cpp file name extensions and some with .h extensions (there are many other extensions but these are the ones I come across most often). So what are the advantages or disadvantages of using many source files? And how do we go about doing this?

To Be Written

Other points

The [wyoGuide](#) by Otto Wyss covers many more points for making your application look and act in a professional manner. It is well worth downloading and reading. Then perhaps use it to add some more features to the sample application.

Sample Application Part 4 – Making It Professional

This section is down to you, consider it homework if you like. Your assignment is to read through Otto Wyss' wyoGuide then implement some or all of the points to make our application more professional. Here is a list of points you might want to try implementing.

Chapter 2 – Basic Frame Layout

Try writing code that sets the main frame to a reasonable default size. You could then try implementing code which allows the user to save and load a preferred size and position. These could be saved in a config file our application already created one that is held in the pointer `m_fileConfig`.

You could alter the title bar to display more information such as the application name and the name of the currently opened file or Untitled.html if the current file has no name.

You could also alter the status bar to contain a new field showing the current file name and more feedback information.

Chapter 6 - 'Preferences'

Try to add a new menu item that produces a preferences dialog. This dialog could allow the user to show/hide the status and toolbars. It could contain options to allow the user to decide whether the splashscreen and tip box should be displayed or not.

On the subject of tips it is also good to keep track of the last tip the user was shown and save it to start in the same place next time the program runs.

You could also add a menu with check items that turn these preferences on or off.

You could also implement loading and saving of these preferences via a config file.

Chapter 11 – 'Miscellaneous'

If you are really adventurous you could try implementing drag and drop to open files.

All of the above points with the exception of drag and drop have been incorporated into the source code for the sample application in the Chapter 11 file.

Basic wxDev-C++ Related FAQs

Q. Why does my program look different when I compile it?

- A. If the differences are confined to the placement of controls, size of controls or other minor differences then there are two reasons.
The first is that wxDev-C++ is written using Delphi not wxWidgets, therefore the look of the controls may not correspond exactly.
The second reason is that wxDev-C++ is a work in hand. Features are still being added and improved, at some later date these differences may be worked on and corrected. If it is a big problem you can file a bug report at sourceforge to alert the developers.
If the problem is that your program doesn't look like an XP program read the next FAQ

Q. Why does my program not look like a Windows XP program when I compile it?

- A. If your program looks like a Windows XP program in the visual designer but when you compile and run it, it looks like a Windows 9x program then the cure is simple.
In wxDev-C++ go to the menu Project|Project Options or press [Alt] + [P] to bring up the following dialog.

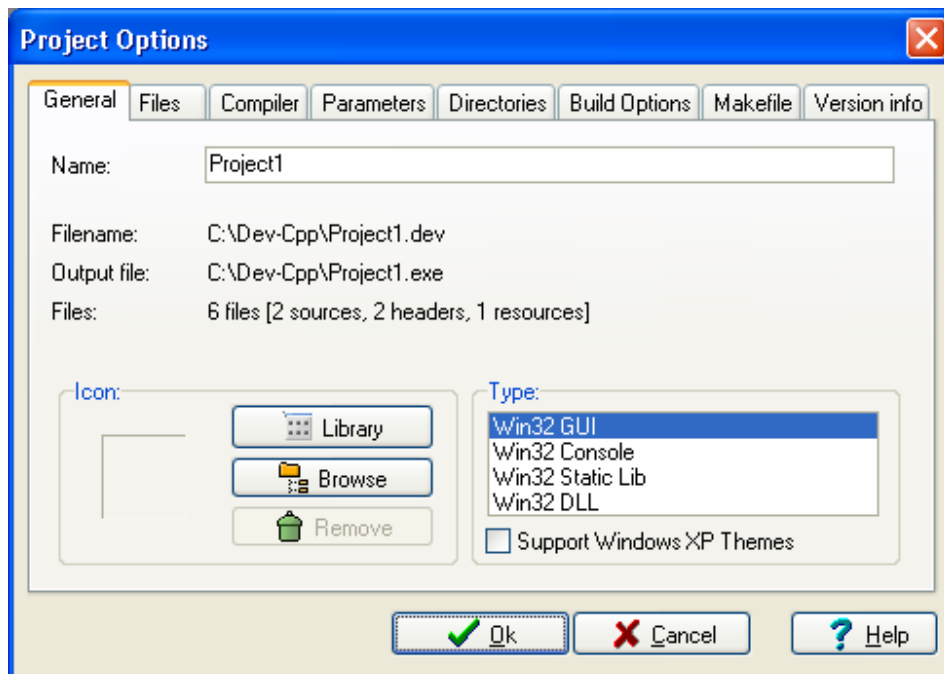


Figure ?? – The Project Options dialog

The tab you want is the one that is already open and just above the buttons is a check box titled 'Support Windows XP Themes', check this. Click on [OK] then press <F9> to recompile and run. You should have a shiny new XP themed program.

Q. wxDev-C++ keeps crashing what's wrong, what can I do?

A. Since wxDev-C++ is pretty stable it is a work in hand and will contain bugs and at times those bugs will crash the IDE. Sometimes it is a problem in wxDev-C++'s code and sometimes it will be due to the configuration of your computer. So what can you do about it?

Firstly don't write to the developers or forums with messages like this "wxDev-C++ crashes, please fix it" or "wxDev-C++ has just crashed what is wrong with it?". The developers are only human (yes it is true!) they can not guess from messages like that what has gone wrong.

The correct procedure is to file a bug report. When the IDE crashes it will generally produce the following dialog.

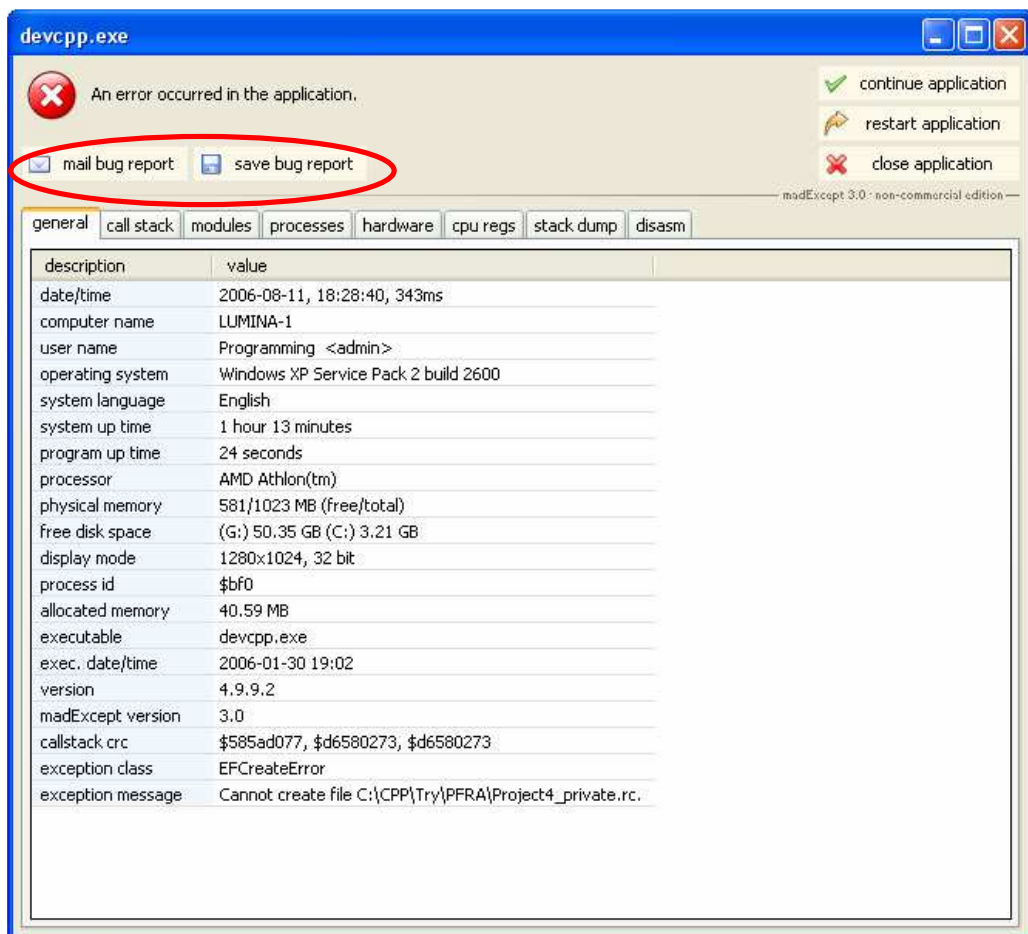


Figure x.x – wxDev-C++ error dialog

The section of most interest to us is circled in red.

The following steps will vary depending on whether you choose to save the bug report first or just mail it. Steps 1 – 2 are the same for both. Step 3 is only if you have chosen to save a bug report.

So choose either

Save bug report
Or Mail bug report

Then follow the next steps

Step 1 In the dialog enter your contact information. This allows the developers to contact you if they need further information or to inform you about possible solution and fixes.

Click [Continue].

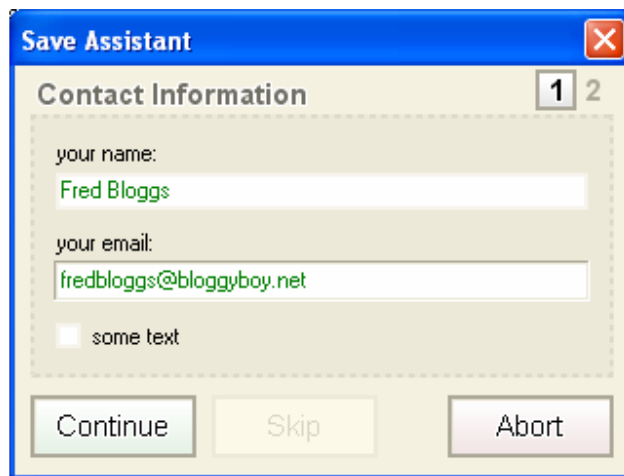
The image shows a Windows-style dialog box titled "Save Assistant". It has a blue title bar with a close button (X) in the top right corner. The main area is light beige and contains a section titled "Contact Information" with a progress indicator showing "1" of "2" steps. Below the title, there are two text input fields. The first is labeled "your name:" and contains the text "Fred Bloggs". The second is labeled "your email:" and contains the text "fredbloggs@bloggyboy.net". Below these fields is a checkbox labeled "some text" which is currently unchecked. At the bottom of the dialog, there are three buttons: "Continue", "Skip", and "Abort". A red circle is drawn around the "Continue" button.

Figure x.x - Step 1, Entering your contact information

Step 2 Although you can skip this step it is not very helpful. So enter everything you did leading up to the crash. Enter as much detail as possible here as this will allow the developers to try to reconstruct the steps that lead to this situation.

Click [Continue]

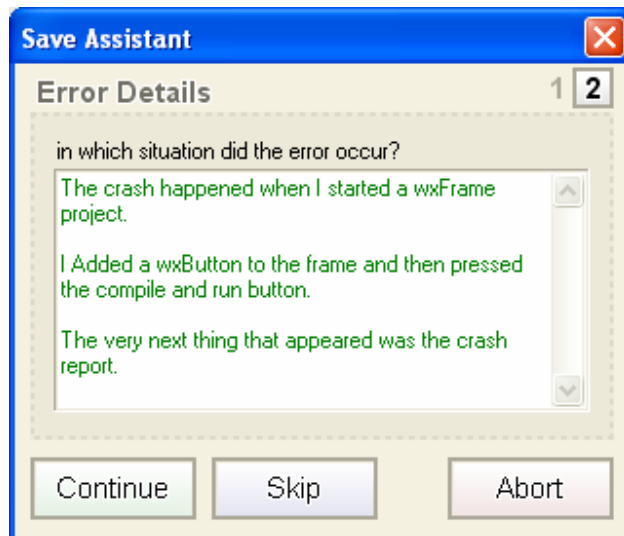


Figure x.x - Step 2, Reporting the steps leading to the crash

Step 3 If you chose to save the bug report then the next dialog you will see is a save dialog. Save the bug report under any name you wish.

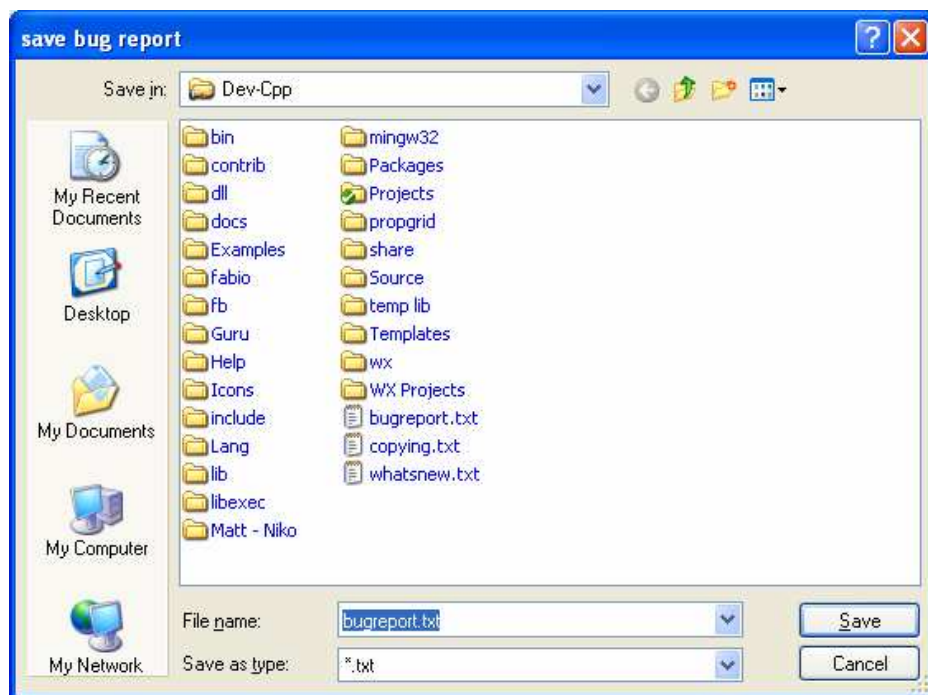


Figure x.x – Step 3, Saving a bug report.

Click [Save]

You will be returned to the crash dialog.

Click [Mail bug report]

Steps 1 and 2 will already be filled, this will be evident by the continue button being coloured dark green.

Continue on to the final stage.

Step 4 The final stage is to send a screenshot of your desktop and the state of wxDev-C++, this is generated automatically by the report generator. This can give the developers useful information about the state of your system and the IDE.

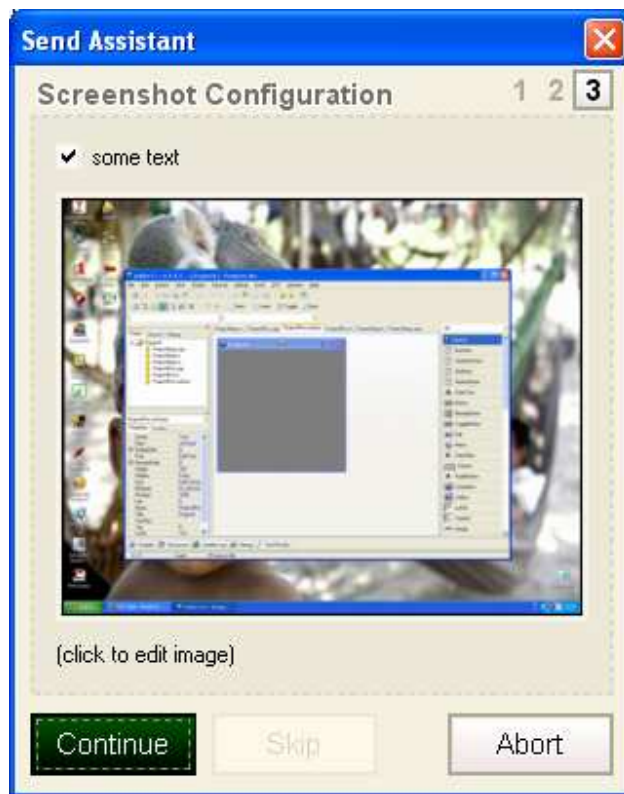


Figure x.x – Screenshot of your desktop and wxDev-C++ running.

Now the report generator will contact your default email client to create an email to the developers.

Step 5 Add any extra information in this email, but nothing rude, remember they are working hard to develop this and provide it free of charge. Finally send the email.

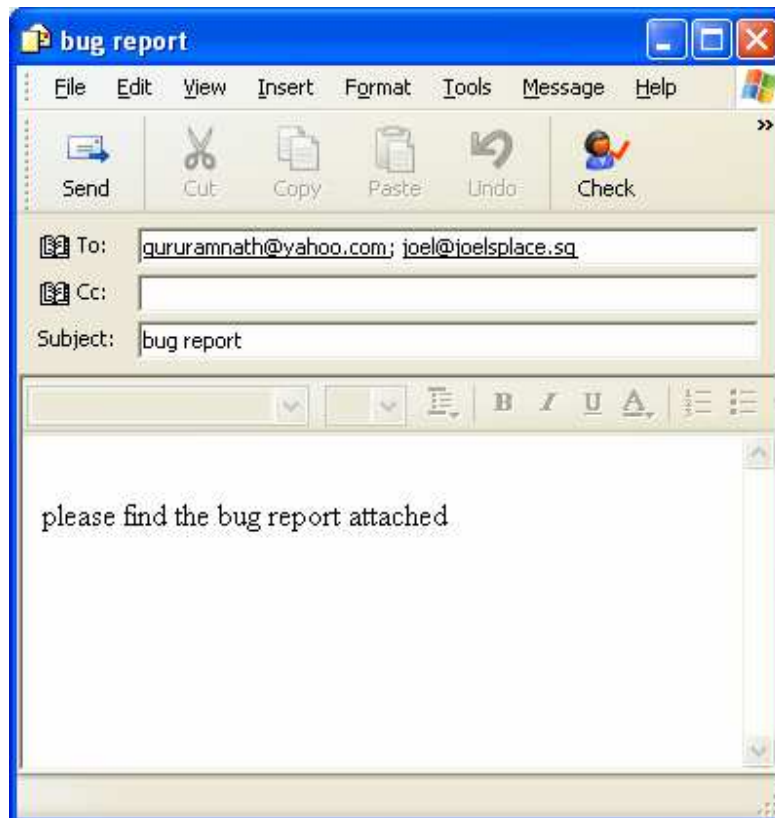


Figure x.x – Email carrying the completed bug report.

Q. I started wxDev-C++ then selected File|New|New wxFrame. Three files Frm1h, Frm1.cpp and Frm1.wxform were created, but when I pressed compile nothing happens. If I press Ctrl+11 a dialog that says ‘Rebuilding...’ appears and nothing happens. What is wrong?

A. Nothing is wrong. The New wxFrame options just adds a new frame to your project. If you haven’t begun a wxWidgets project then there is nothing for the compiler to compile therefore it wont do anything.

Q. I try to compile my project but get lots of errors like the following:

**My ProjectFrm.h:33: error: expected unqualified-id before numeric constant
My ProjectFrm.h:33: error: expected `,' or `;' before numeric constant**

What is the problem?

A. The most usual cause of this is that when you created a new project you included spaces in the project name. If unnoticed this can result in the generated class names containing spaces which is not legal in C++. Spaces in the project’s filenames can also create problems since the make program which is used to call the compiler can interpret the space as the end of the filename. (The problem with spaces in the filenames is reported to be fixed).

- Q. I disabled code completion, but the visual designer keeps complaining, why does the designer need code completion enabled?**
- A. When you use the designer to add event function to your project, the designer needs to determine the best place to insert this function and check that a function with the same name does not already exist. If you don't intend to add events you don't need code completion enabled.
- Q. I have installed a new version of wxDev-C++ or a new version of the wxWidgets library. Now when I try to compile a previously working project I get this error "cannot find -lwxmsw25". What is wrong?**
- A. The problem that is being reported is that the linker cannot find one of the library files. In this case it is the base wxWidgets file. This problem generally arises due to the naming convention of the wxWidgets libraries. These are named 'l' for library 'wx' for wxWidgets, then the platform 'msw', finally the library version number in this case 2.5.x. It is the last part that causes problems. Your project has been linked to the 2.5.x library but if you have updated you may now be using 2.6.x or 2.7.x and so on. So how to fix the problem?

Open the Project Options dialog via Project|Project Options. Then select the tab labelled "Additional Command-line options". In the text control under the label Linker look for any options that contain a 25 and alter this to 26 or whatever version of library you are using.

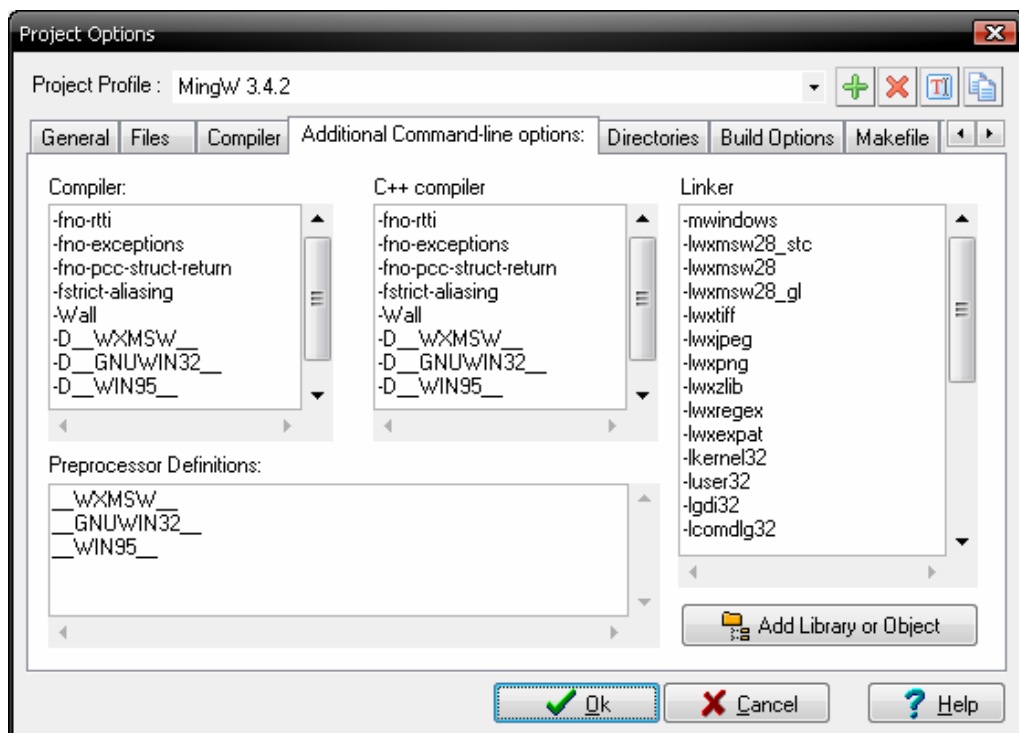


Figure x.x – The Project Options dialog showing wxWidgets 2.6.x library files

- Q. I have installed a new version of wxDev-C++ or a new version of the wxWidgets library. Everything compiles fine, but when I try to run the program I get a library mismatch error dialog. What is wrong?**
- A. This problem occurs because some parts of the program are compiled with the old library and other parts are compiled with the new library. As a result you will see an error dialog like the following.



Figure x.x – The library mismatch dialog

The answer is simple. Go to the menu option Execute|Rebuild All or press Ctrl + F11. This will recompile all the source files in your application with the new library.

- Q. I altered a line in the wxWidgets setup.h file to enable a feature I need. Now when I build my program I get a library mismatch dialog. What did I do wrong?**
- A. This problem occurs because the wxWidgets library has been compiled with the settings previously in the setup.h file. However because you have altered the settings in setup.h they no longer match those within the library. As a result you will see an error dialog like the following.

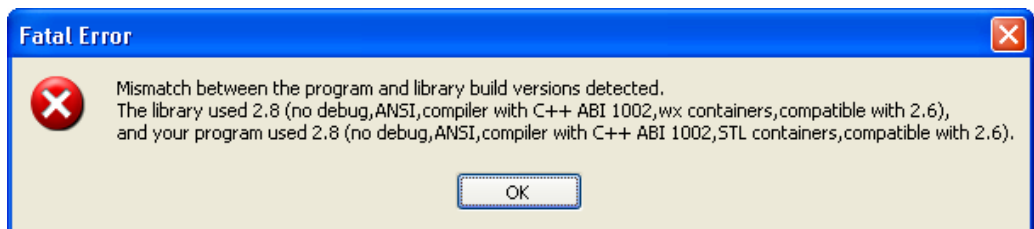


Figure x.x – The library mismatch dialog

There are two solutions, the first is if you can manage without the feature simply change the settings in setup.h back to their previous settings.

The second option is if you need the features. Then you will need to recompile the wxWidgets libraries using the settings you need.

Q. I try to compile my wxWidgets project but I keep getting the error “[Resource error] can’t open cursor file ‘wx/msw/hand.cur’: No such file or directory. What am I doing wrong?”

A. Firstly don’t worry this is an ongoing problem that raises its head every now and then. This should be fixed in the latest versions of wxDev-C++ but if you do run into this problem here is how to fix it. Open the Project Options dialog via Project|Project Options. Go to the Directories tab and under Resource Directories add the include path to your copy of wxDev-C++ this will be either ‘C:\Dev-Cpp\include’ if you installed to the default directory or whatever path you chose to install to plus ‘\include’.

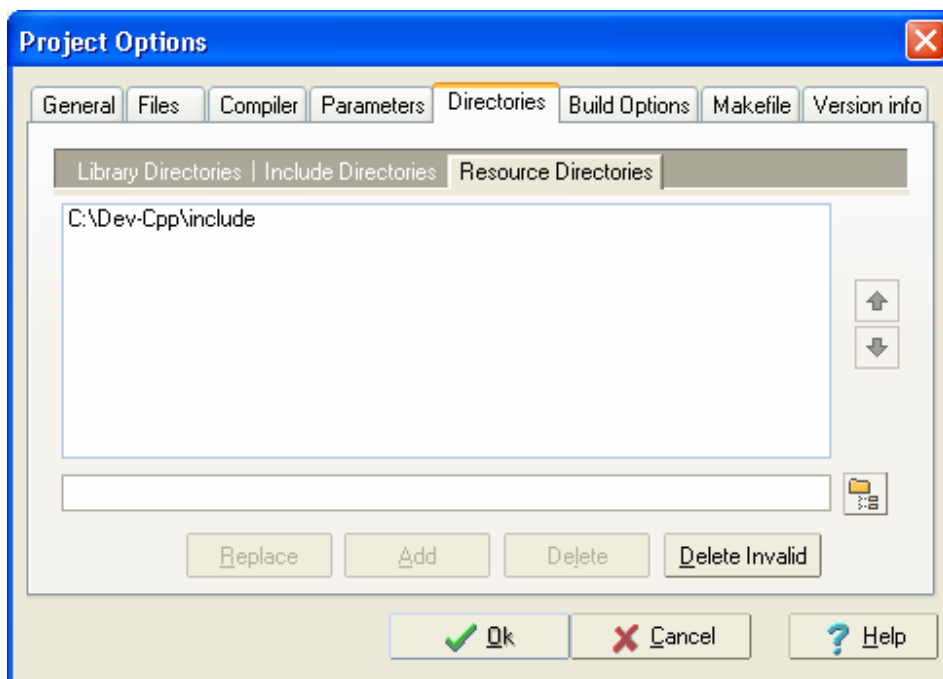


Figure x.x – Project Options dialog showing the resource include path.

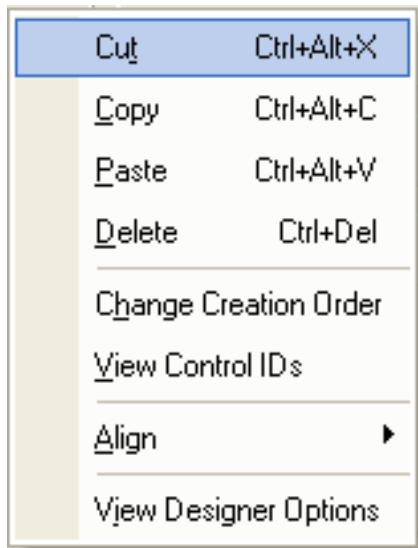
Q. I tried to compile my project but I get the error

**In file included from MyProject_private.rc
wx/msw/wx.rc: No such file or directory.**

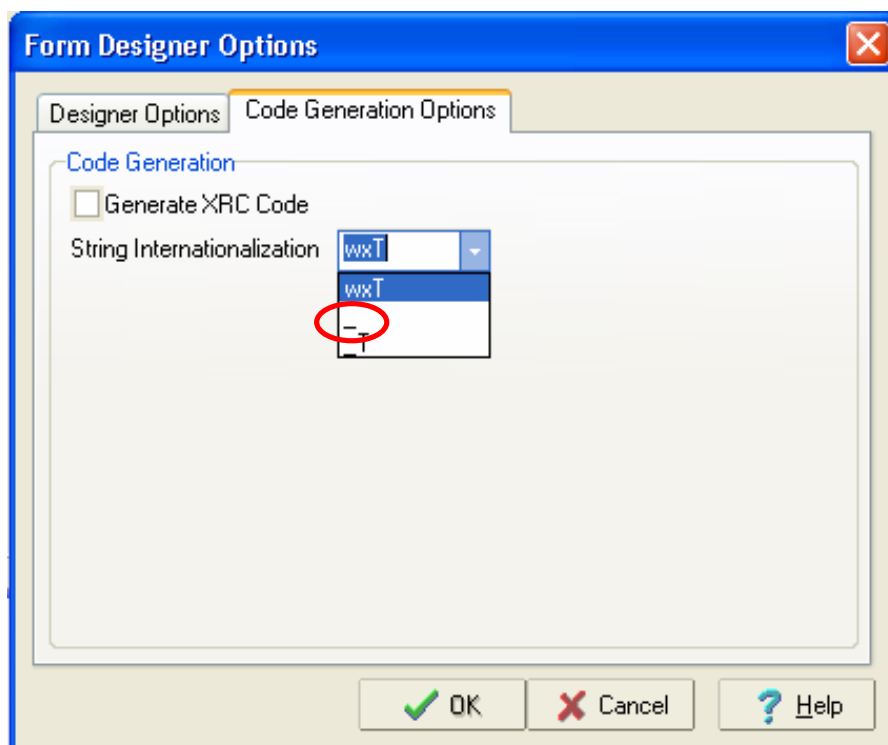
A. This problem is related to the proceeding problem and the answer is the same.

Q. wxDev-C++ adds wxT() around my strings, however I want my strings to be translatable. How do I get wxDev-C++ to add _() around my strings.

A. In wxDev-C++ 6.9 onwards right click on the designer form. From the following menu choose “View Designer Options”.



From the following dialog choose the tab “Code Generation Options”, From the drop down box labelled “String Internationalization” Choose the option _().



Q. I moved my project files (or downloaded the source code examples) and now I’m getting an error like

mingw32-make.exe: * No rule to make target
`../Src/Chapter8/Default_Profile/HTMLEdit_private.rc', needed by
`../Src/Chapter8/Default_Profile/HTMLEdit_private.res'**

- A. The reason for this is that wxDev-C++ holds the full path to the projects resources in the projects .dev file. The work around is simple.

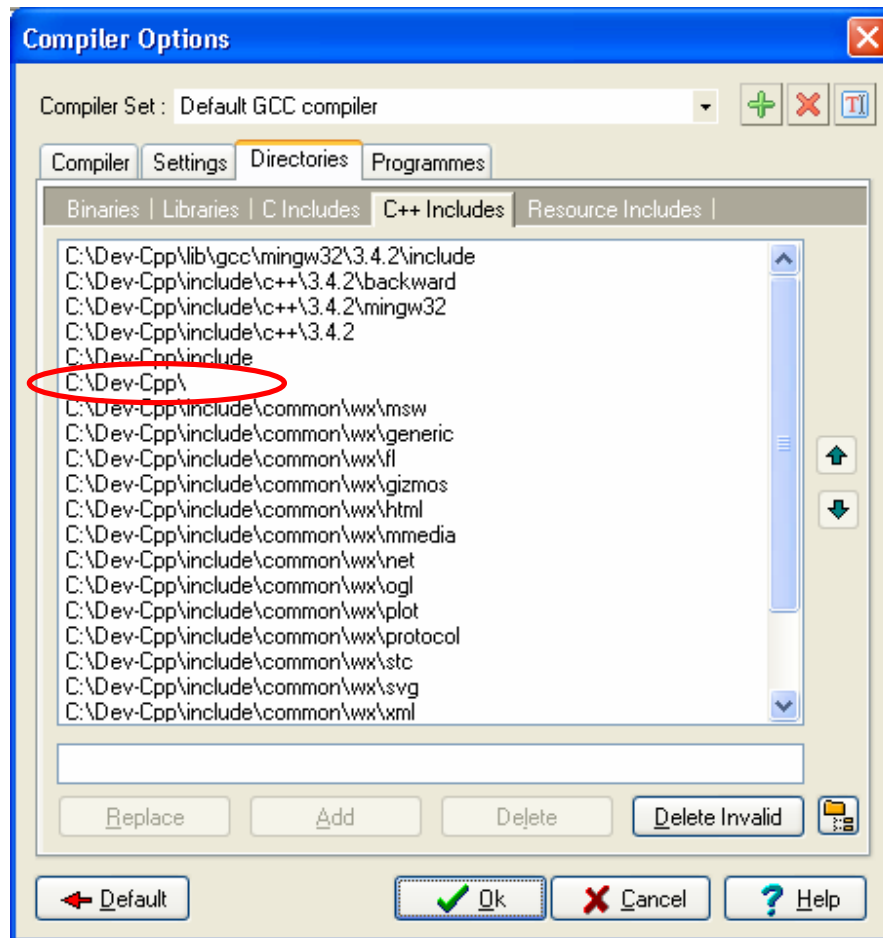
Open the Project Option dialog via Project|Project Options.
Click on the [OK] button

This will cause the file paths to be regenerated.

- Q. I am using wxDev-C++ Version 6.10 or 6.10.1 and the class browser does not seem to be working. What can I do?**

- A. If your project located under the installation directory (e.g. C:\Program Files\Dev-Cpp\My Project)? If so then follow these steps.

Go to Tools|Compiler Options.
In the Compiler Options dialog goto the Directories tab
Under the Directories tab choose the C++ Includes tab
Removing the installation directory e.g. C:\Program Files\Dev-Cpp or
C:\Dev-Cpp from the list of included directories
Then close and restart wxDev-C++



Alternatively you can move your project to a different directory e.g. C:\My Documents\My Project

- Q. I am using wxDev-C++ with the Microsoft compiler. At link time I get the error <filename>.obj : error LINK2001 : unresolved external symbol __pexit. What is wrong?**
- A. To solve this go to Project|Project Options. In the resulting dialog goto Compiler tab, then Code Generation. Set the option enable __pexit function call to False.
- Q. I want to create large frames but I cannot drag the frame beyond the boundaries of the designer or access all of the frame.**
- A. To make the frame the size that you want, enter the height and width in the Property Inspector. To access parts of the frame hidden by the IDE you need to move the frame altering the top and left dimensions in the Property Inspector to minus figures.

Q. I have one or more floating yellow boxes containing function definitions that reappear every time I load my project. Even when I minimise wxDev-C++ they are still there. How can I get rid of them?

A. The problem occurs because the text cursor is inside the parameter section of a function. Normally when you switch tab the pop up box disappears. However the cursor positions are saved and when a project is reloaded the popup boxes can appear if the cursor on a tab is within a function parameter.

Q. I have one or more floating yellow boxes containing function definitions that reappear every time I load my project. Even when I minimise wxDev-C++ they are still there. How can I get rid of them?

1 / 2 wxStaticText ()

A.

Q. Where can I go for more help?

A. The best place to ask for wxDev-C++ related help is on the [wxForum](#). There are

Figure x.x – The popup boxes while I am writing this item

```
63  
64 wxStaticText2 = new wxStaticText(this, ID_W  
65 wxStaticText2->SetFont(wxFont(8, wxSWISS, w  
66
```

Figure x.x – The cursor was positioned where the red circle is

Until a fix is made the solution is to select each tab that contains code and move the cursor to a new position, preferably a blank line.

Q. Where can I go for more help?

A. The best place to ask for wxDev-C++ related help is on the [wxForum](#). There are special sub conferences devoted to using [wxDev-C++](#), [writing C++ programs](#), using wxWidgets [addons](#) and much more.

Another resource for wxWidgets related problems is the [wxWiki](#). More resources can be located in the Resources Section.

Part 3 – Advanced Development with wxWidgets and wxDevC++

Chapter 13 - Creating and using other controls

The Simple Way

The Complex Way

Chapter 14 – Working with other frameworks

OpenGL

SDL

Part 4 – Going Beyond the Boundaries of this Book

Alternatives

wxFormDesigner

Code::Blocks

DialogBlocks

Resources

C Programming

[Thinking in C](http://mindview.net/CDs/ThinkingInC/beta3) - <http://mindview.net/CDs/ThinkingInC/beta3>

C++ Programming

[Thinking in C++ Volume 1+2 by Bruce Eckel](http://mindview.net/Books) - <http://mindview.net/Books>

<http://www.freeprogrammingresources.com/cppbooks.html>

C++ Beginners tutorial: <http://www.cplusplus.com/doc/tutorial/>

wxWidgets Programming

[wxForum](http://wxforum.shadonet.com/index.php) - <http://wxforum.shadonet.com/index.php>

[C++ Programming](http://wxforum.shadonet.com/viewforum.php?f=1&sid=f3c4ab06a21f8248456041044680ff5c) -

<http://wxforum.shadonet.com/viewforum.php?f=1&sid=f3c4ab06a21f8248456041044680ff5c>

[wxDev-C++](#) -

<http://wxforum.shadonet.com/viewforum.php?f=28&sid=f3c4ab06a21f8248456041044680ff5c>

[wxCode](#) -

<http://wxforum.shadonet.com/viewforum.php?f=30&sid=f3c4ab06a21f8248456041044680ff5c>

[wxWiki](#) - http://www.wxwidgets.org/wiki/index.php/Main_Page

Program Design

[wyoGuide](#) - <http://wyoguide.sourceforge.net/guidelines/content.html> - by Otto Wyss

Articles about wxDev-C++

[Cross-platform GUI development with wxWidgets](#) -

http://www.developer.net.au/Crossplatform_GUI_development_with_wxWidgets.htm - by Daniel Winter

Appendix

Appendix B – C/C++ Keywords

Keyword	C89	C99	C++	Meaning
_Bool		✓		
_Complex		✓		
_Imaginary		✓		
asm			✓	
auto	✓	✓	✓	
bool			✓	
break	✓	✓	✓	
case	✓	✓	✓	
catch			✓	
char	✓	✓	✓	
class			✓	
const	✓	✓	✓	
const_cast			✓	
continue	✓	✓	✓	
default	✓	✓	✓	
delete			✓	
do	✓	✓	✓	
double	✓	✓	✓	
dynamic_cast			✓	
else	✓	✓	✓	
enum	✓	✓	✓	
explicit			✓	
export			✓	
extern	✓	✓	✓	
false			✓	
float	✓	✓	✓	
for	✓	✓	✓	
friend			✓	
goto	✓	✓	✓	
if	✓	✓	✓	
inline		✓	✓	
int	✓	✓	✓	
long	✓	✓	✓	
mutable			✓	
namespace			✓	

new			✓
operator			✓
private			✓
protected			✓
public			✓
register	✓	✓	✓
reinterpret_cast			✓
restricted		✓	
return	✓	✓	✓
short	✓	✓	✓
signed	✓	✓	✓
sizeof	✓	✓	✓
static	✓	✓	✓
static_cast			✓
struct	✓	✓	✓
switch	✓	✓	✓
template			✓
this			✓
throw			✓
true			✓
try			✓
typedef	✓	✓	✓
typeid			✓
union	✓	✓	✓
unsigned	✓	✓	✓
using			✓
virtual			✓
void	✓	✓	✓
volatile	✓	✓	✓
wchar_t			✓
while	✓	✓	✓

Appendix C – wxDev-C++ event names and wxWidgets equivalents

Event Name	Example Control	Event Occurs When
On3StateImageClick	TreeCtrl	
OnActivate	Frame	The frame receives or loses focus, usually this will be indicated by the caption changing colour. Event name - 'wxActivateEvent'.
OnActivateApp	Frame	When the application receives or loses focus, usually by one of its frame receiving or losing focus. Event name - 'wxActivateEvent'.
OnAutoCompSelection	StyledTextCtrl	
OnBeginDrag	ListCtrl	
OnBeginLabelEdit	ListCtrl	
OnBeginRDrag	ListCtrl	
OnCacheHint	ListCtrl	
OnCallTipClick	StyledTextCtrl	
OnChange	StyledTextCtrl	
OnChar	Frame	A keypress occurs. The event translates the key pressed into its actual value for example if the 'A' is pressed and shift is not then the event will hold 'a'. For more info see 'wxKeyEvent'.
OnCharAdded	StyledTextCtrl	
OnCheckListBox	CheckListBox	
OnClick	Button	
OnClickUrl	TextCtrl	
OnClose	Frame	A user tries to close the frame, for example by clicking on the 'X' button in the caption. This event is helpful if you want to carry out some process like saving information before the frame closes. See 'wxCloseEvent' for further details.
OnColBeginDrag	ListCtrl	
OnColDragging	ListCtrl	
OnColEndDrag	ListCtrl	
OnColLeftClick	ListCtrl	
OnColRightClick	ListCtrl	

OnColourChange	Frame	A system colour is changed for example through the control panel. See 'wxSysColourChangedEvent' for more details.
OnDateChanged	DatePickerCtrl	
OnDay	CalendarCtrl	
OnDeleteAllItems	ListCtrl	
OnDeleteItem	ListBox	
OnDeselectItem	ListCtrl	
OnDeselected	ListCtrl	
OnDialupConnected	DialUpManager	
OnDialupDisConnected	DialUpManager	
OnDoDrop	StyledTextCtrl	
OnDoubleClick	SplitterWindow	
OnDoubleClicked	ListBox	
OnDown	SpinButton	
OnDragOver	StyledTextCtrl	
OnDropFiles	Frame	Files have been drag onto a window and dropped. See 'wxDropFilesEvent' for more details.
OnDwellEnd	StyledTextCtrl	
OnDwellStart	StyledTextCtrl	
OnEndDrag	TreeCtrl	
OnEndLabelEdit	ListCtrl	
OnEndSession	Frame	The whole application is ending. See 'wxCloseEvent' for further details.
OnEnter	TextCtrl	
OnEnterWindow	Frame	When the mouse cursor first enters the window. See 'wxMouseEvent' for more details.
OnEraseBackground	Frame	Just before a window paint event. See 'wxEraseEvent' for more details.
OnGetInfo	TreeCtrl	
OnHotspotClick	StyledTextCtrl	
OnHotspotDClick	StyledTextCtrl	
OnHyperLink	HyperlinkCtrl	
OnIdle	Frame	The system becomes idle, normally only one such event is sent each time the system idles. See 'wxIdleEvent' for further details.

OnInitDialog	Frame	The window is being initialised. See 'wxInitDialogEvent' for more details.
OnInsertItem	ListCtrl	
OnItemActivated	ListCtrl	
OnItemCollapsed	TreeCtrl	
OnItemCollapsing	TreeCtrl	
OnItemDeSelected	RichTextCtrl	
OnItemExpanded	TreeCtrl	
OnItemExpanding	TreeCtrl	
OnItemFocused	ListCtrl	
OnItemGetTooltip	TreeCtrl	
OnItemMClick	TreeCtrl	
OnItemMenu	TreeCtrl	
OnItemRClick	TreeCtrl	
OnItemSelected	RichTextCtrl	
OnJoyButtonDown	Frame	wxJoyButtonDown
OnJoyButtonUp	Frame	
OnJoyMove	Frame	
OnJoyZMove	Frame	
OnKey	StyledTextCtrl	
OnKeyDown	Frame	A key is pressed down. If the key is held down then this event will be sent repeatedly. The keys value is not translated. See 'wxKeyEvent' for more details.
OnKeyUp	Frame	A previously depressed key is released. Unlike OnKeyDown, there is only one OnKeyUp event. The keys value is not translated. See 'wxKeyEvent' for more details.
OnKillFocus	Frame	
OnLabelLeftClick	Grid	
OnLabelLeftDoubleClick	Grid	
OnLabelRightClick	Grid	
OnLabelRightDoubleClick	Grid	
OnLeaveWindow	Frame	
OnLeftDClick	Frame	
OnLeftDown	Frame	
OnLeftUp	Frame	
OnMacroRecord	StyledTextCtrl	
OnMarginClick	StyledTextCtrl	
OnMaxLen	TextCtrl	
OnMediaFinished	MediaCtrl	

OnMediaLoaded	MediaCtrl	
OnMediaPause	MediaCtrl	
OnMediaPlay	MediaCtrl	
OnMediaStop	MediaCtrl	
OnMenu	ToolBar	
OnMenuClose	Frame	
OnMenuHighLightAll	Frame	
OnMenuOpen	Frame	
OnMiddleClick	ListCtrl	
OnMiddleDclick	Frame	
OnMiddleDown	Frame	
OnMiddleUP	Frame	
OnModified	StyledTextCtrl	
OnMonth	CalendarCtrl	
OnMouseEvents	Frame	
OnMouseMove	Frame	
OnMouseWheel	Frame	
OnNeedShown	StyledTextCtrl	
OnPageChanged	Notebook	
OnPageChanging	Notebook	
OnPaint	Frame	
OnPainted	StyledTextCtrl	
OnQueryEndSession	Frame	
OnRangeSelect	Grid	
OnReturn	RichTextCtrl	
OnRightClick	ListCtrl	
OnRightDclick	Frame	
OnRightDown	Frame	
OnRightUP	Frame	
OnROModifyAttempt	StyledTextCtrl	
OnRowSize	Grid	
OnSashPosChanged	SplitterWindow	
OnSashPosChanging	SplitterWindow	
OnSavePointLeft	StyledTextCtrl	
OnSavePointReached	StyledTextCtrl	
OnScroll	Slider	
OnScrollbar	ScrollBar	
OnScrollbarBottom	ScrollBar	
OnScrollEnd	ScrollBar	
OnScrollLineDown	ScrollBar	
OnScrollLineUp	ScrollBar	
OnScrollPageDown	ScrollBar	
OnScrollPageUp	ScrollBar	
OnScrollThumbRelease	ScrollBar	
OnScrollThumbtrack	ScrollBar	

OnScrollTop	ScrollBar	
OnScrollWin	Frame	
OnScrollWinBottom	Frame	
OnScrollWinLineDown	Frame	
OnScrollWinLineUp	Frame	
OnScrollWinPageDown	Frame	
OnScrollWinPageUp	Frame	
OnScrollWinThumbRelease	Frame	
OnScrollWinThumbTrack	Frame	
OnScrollWinTop	Frame	
OnSelChanged	TreeCtrl	
OnSelChanging	TreeCtrl	
OnSelectCell	Grid	
OnSelected	wxChoice	
OnSetFocus	Frame	
OnSetInfo	TreeCtrl	
OnSize	Frame	
OnSpinDown	SpinCtrl	
OnSpinUp	SpinCtrl	
OnSplitterDoubleClick	Frame	
OnSplitterSashPosChanged	Frame	
OnSplitterUnSplit	Frame	
OnStartDrag	StyledTextCtrl	
OnSTCUpdateUI	StyledTextCtrl	
OnStyleNeeded	StyledTextCtrl	
OnTextEnter	wxComboBox	
OnTimer	Timer	
OnTool	ToolBar	
OnToolEnter	ToolBar	
OnUnSplit	SplitterWindow	
OnUp	SpinButton	
OnUpdated	TextCtrl	
OnUpdateUI	Most	
OnURIDropped	StyledTextCtrl	
OnUserListSelection	StyledTextCtrl	
OnWeekDayClicked	CalendarCtrl	
OnYear	CalendarCtrl	
OnZoom	StyledTextCtrl	