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

Programming With wxDev-C++

"It's got some quirks but then again don't we all"
NinjaNL

Contents

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 100^0 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.

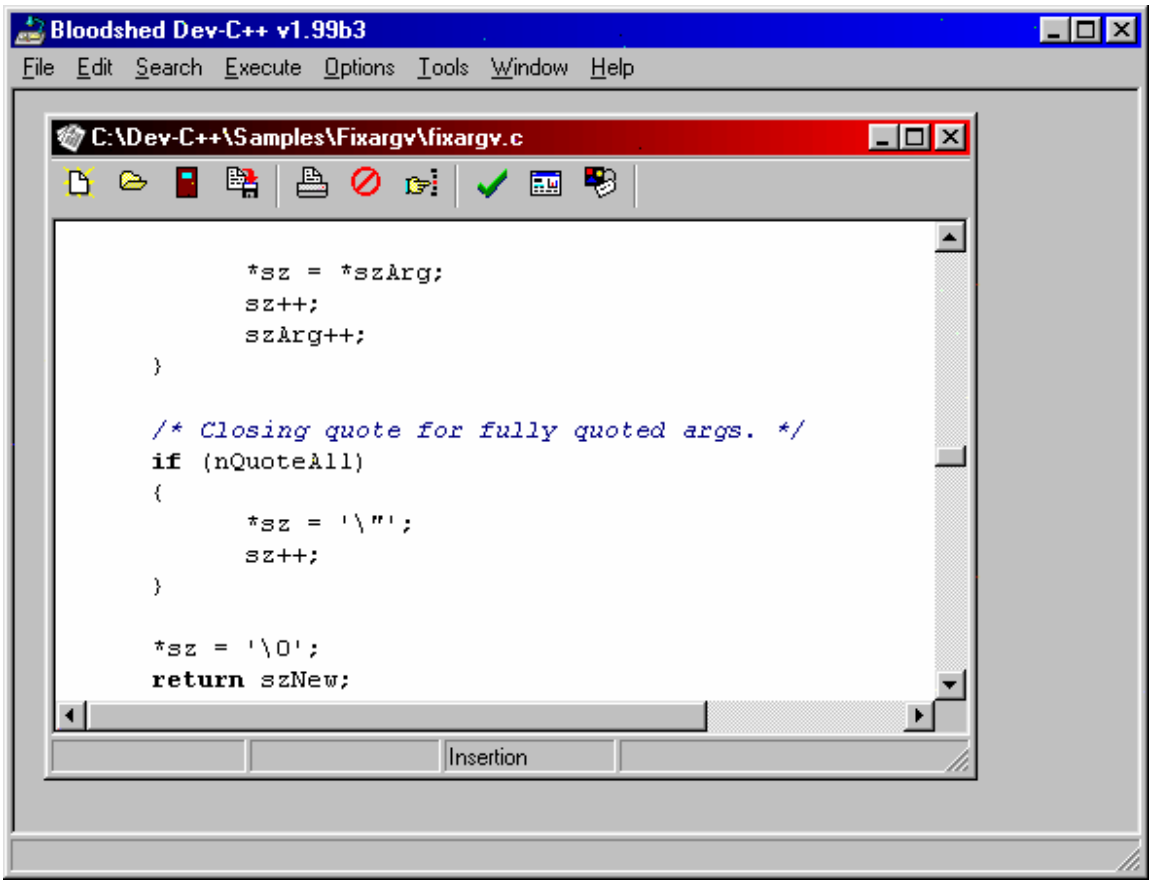


Figure 0.1 – Dev-C++ circa 1999

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.

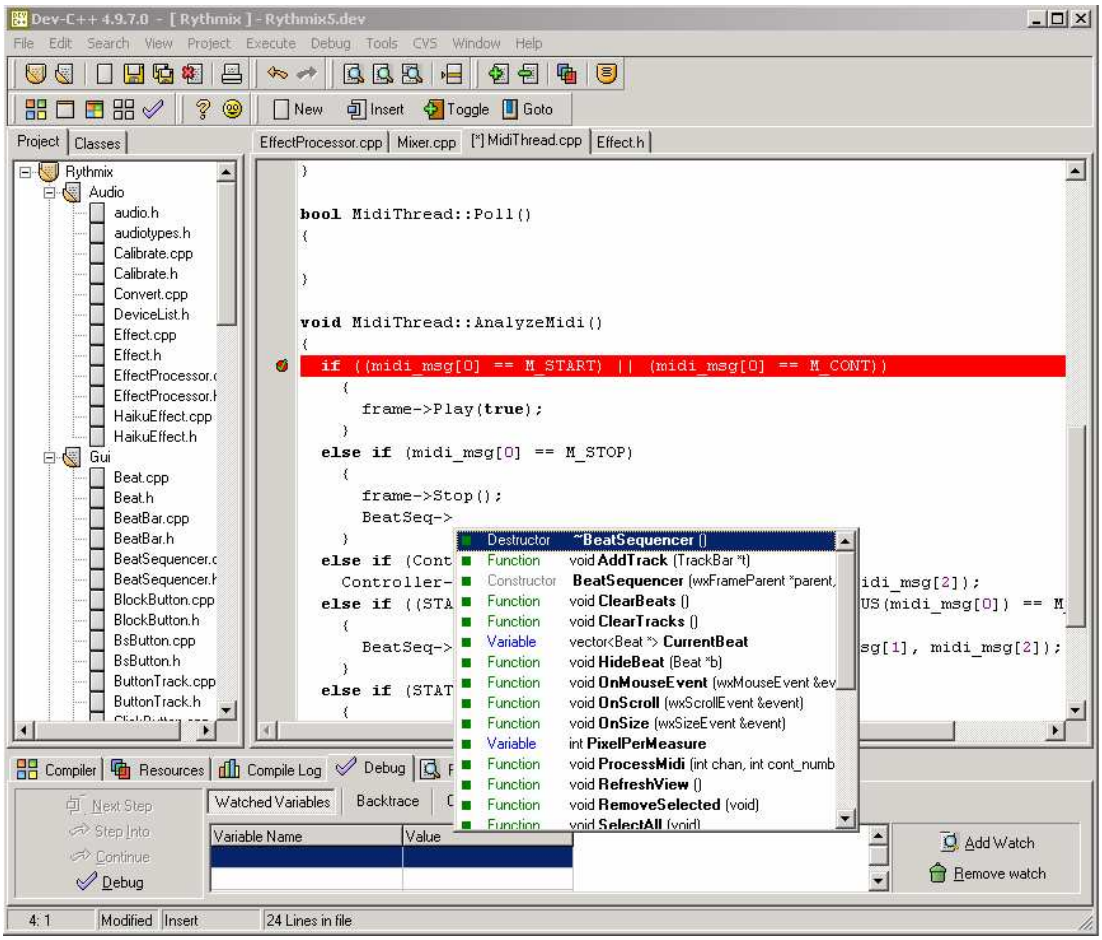


Figure 0.2 – DevC++ in a more recent guise

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 MinGW (Minimalist GNU for Windows). 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 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.

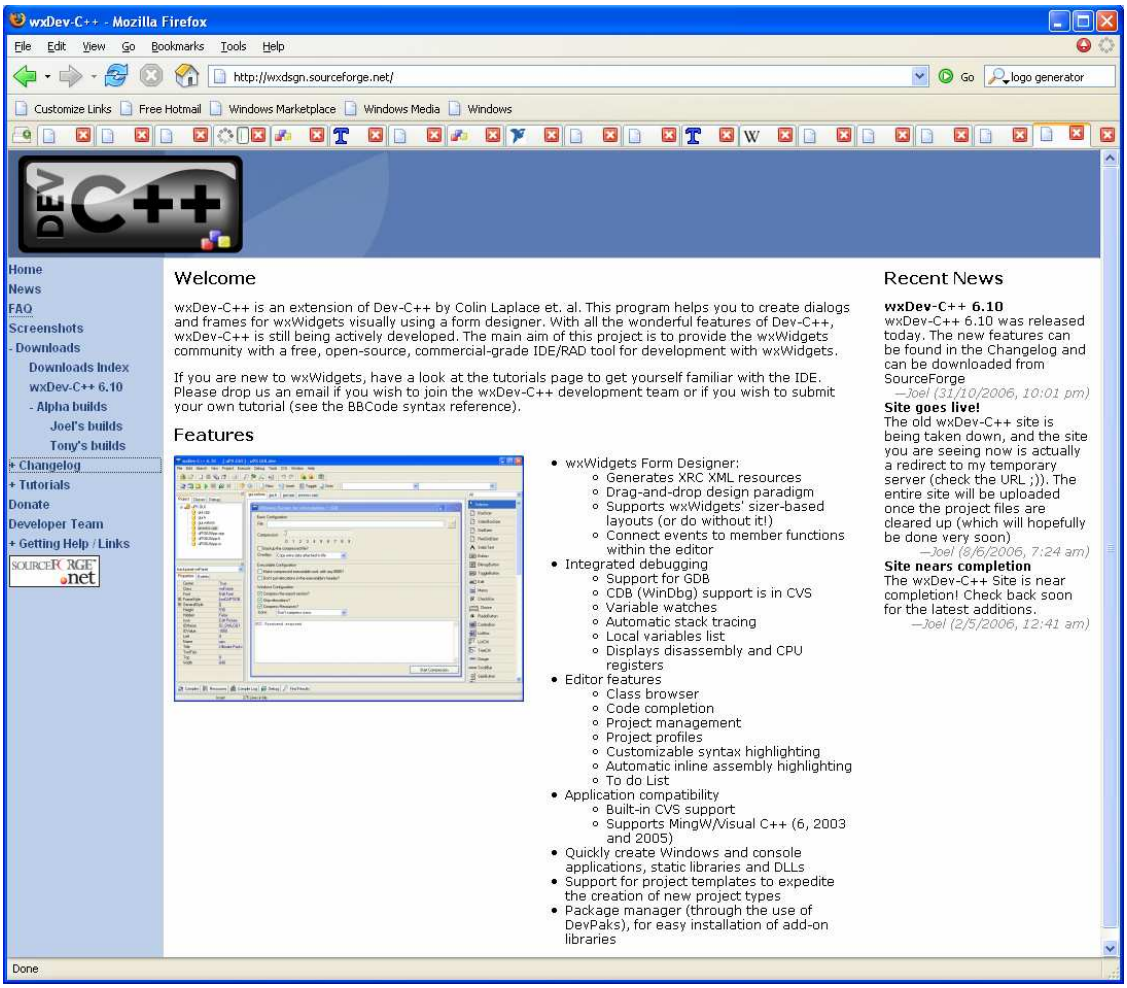


Figure 1.1 – The Official wxDev-C++ website [Accessed 1/11/2006 at 7:55A.M.]

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.

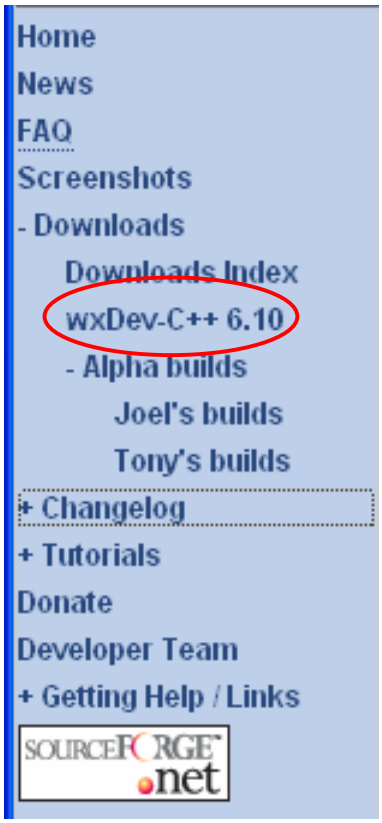


Figure 1.2 – Selecting a version of wxDev-C++

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.

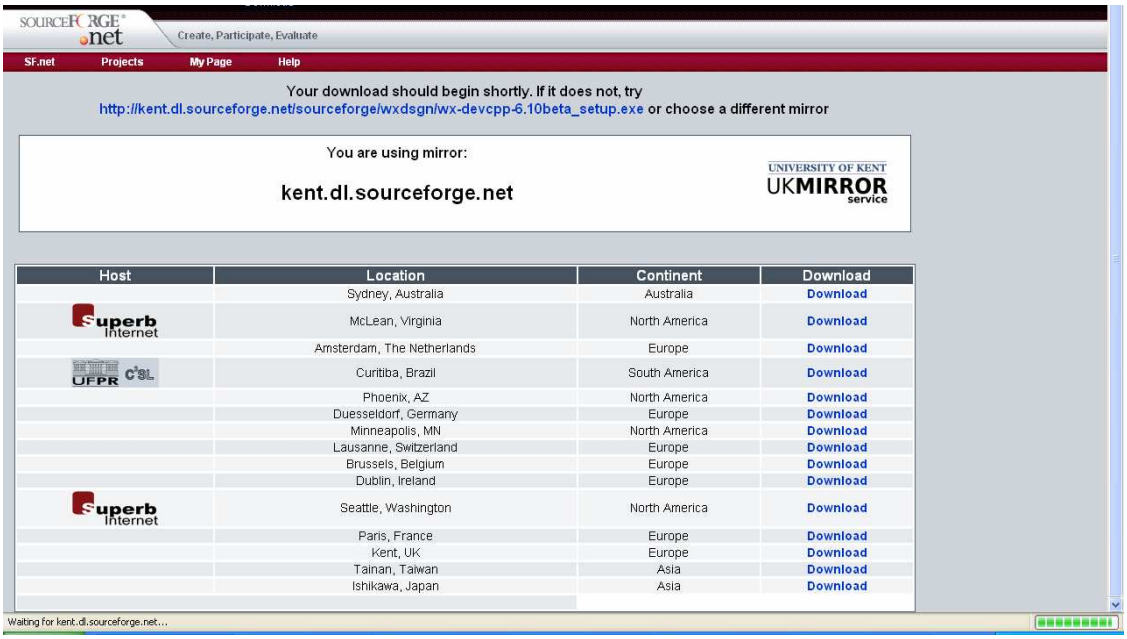


Figure 1.3 – SourceForge download page for wxDev-C++

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.

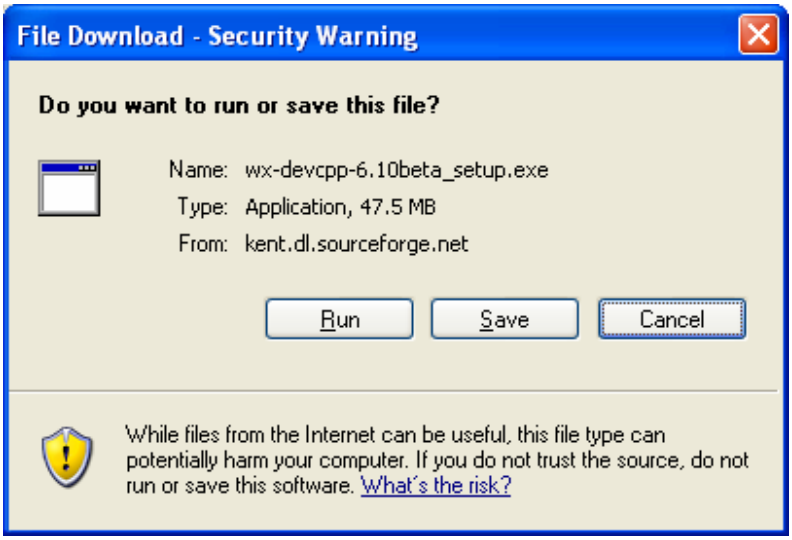


Figure 1.4 – Internet Explorer’s file download option box

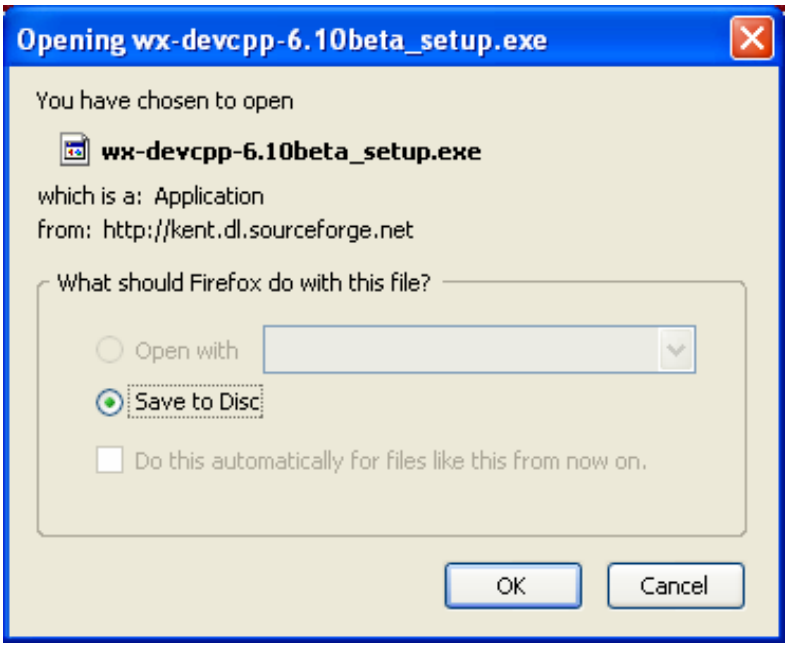


Figure 1.5 – Firefox's file download option box

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.

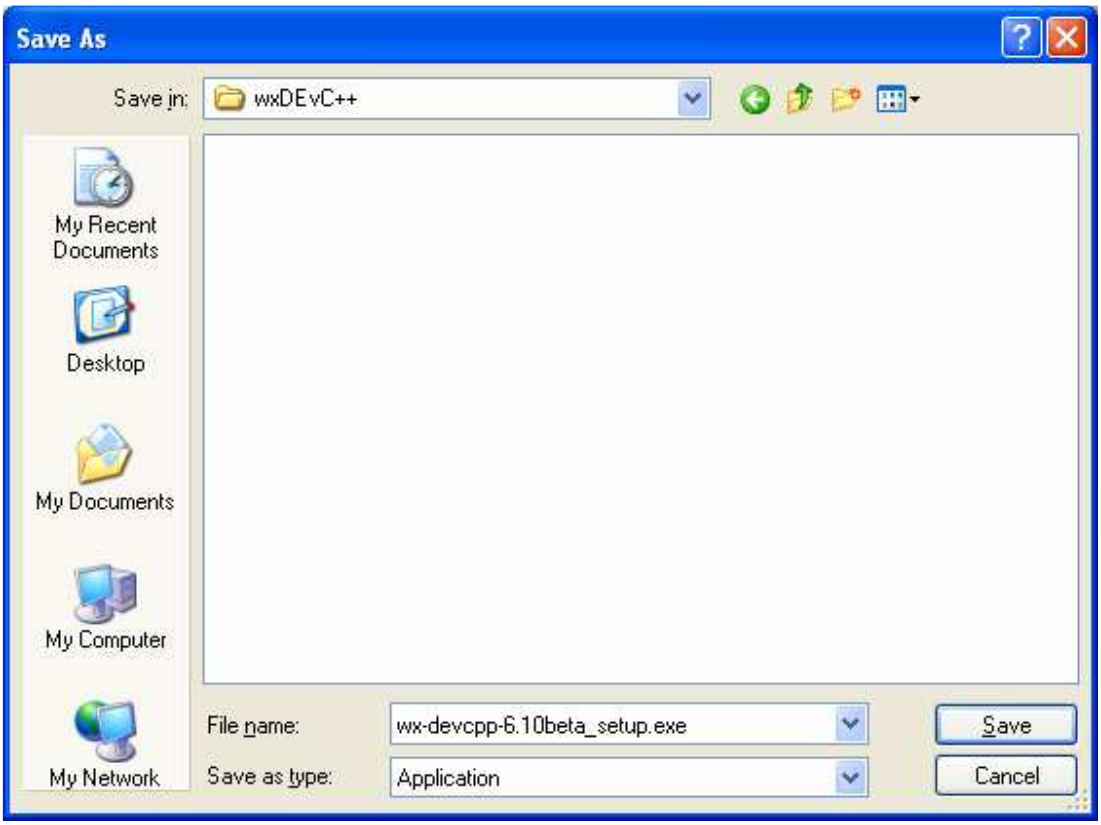


Figure 1.6 – Choosing a location to save the setup files in.

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)

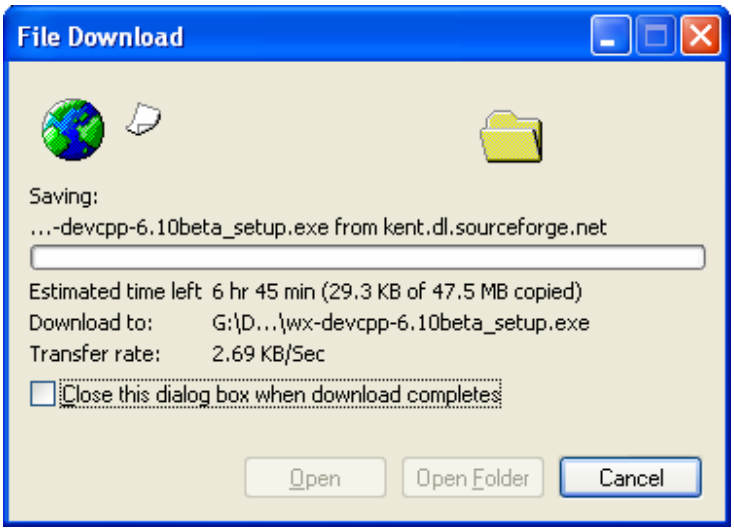


Figure 1.7 – The setup file downloading

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'.

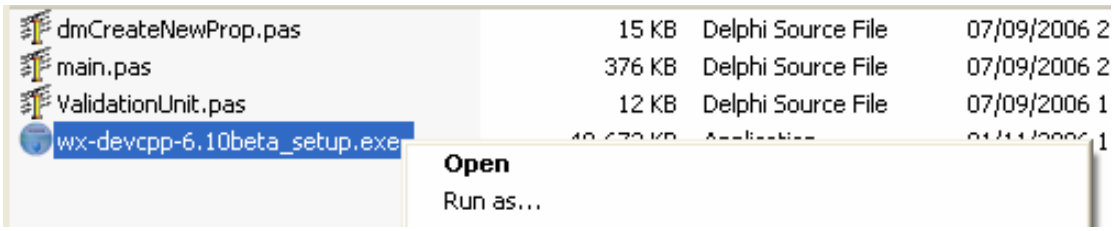


Figure 1.8 – Running the setup program

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.

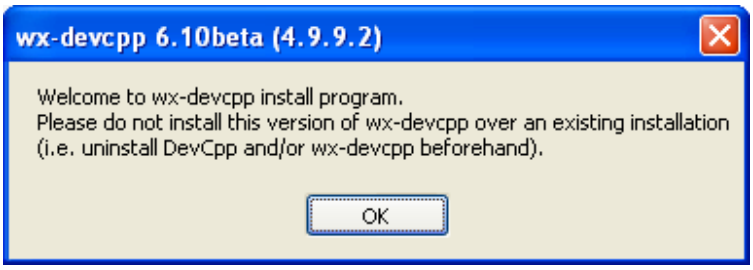


Figure 1.9 – The initial warning message from wxDev-C++ setup

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>]

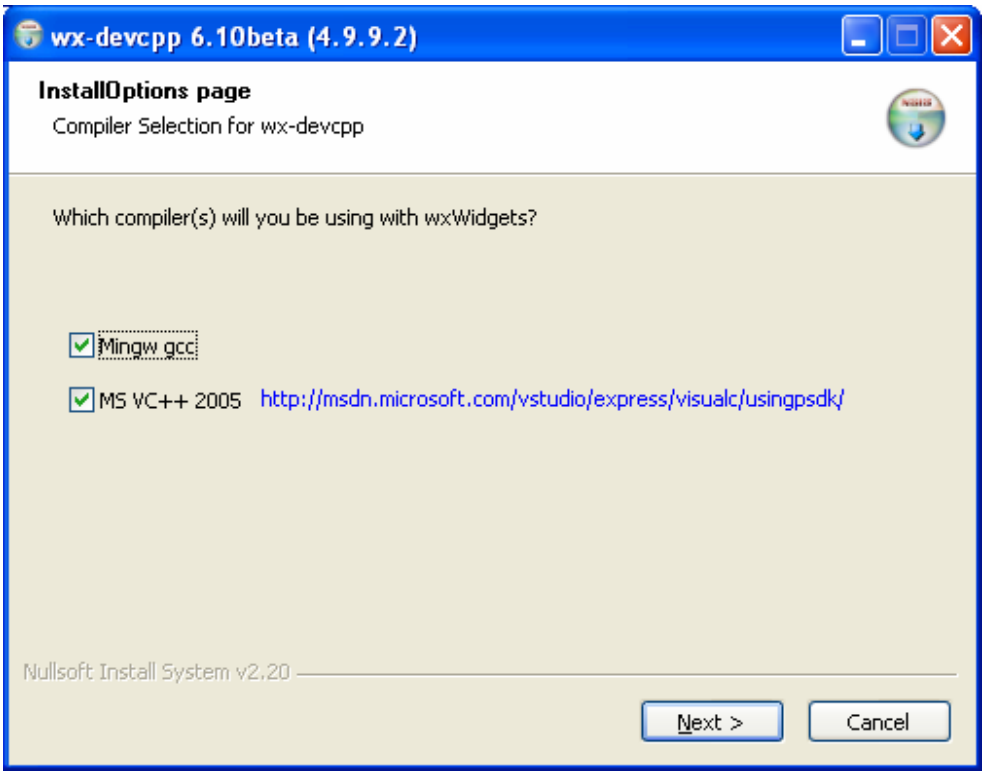


Figure 1.11 – Choosing which compilers you wish to use

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.

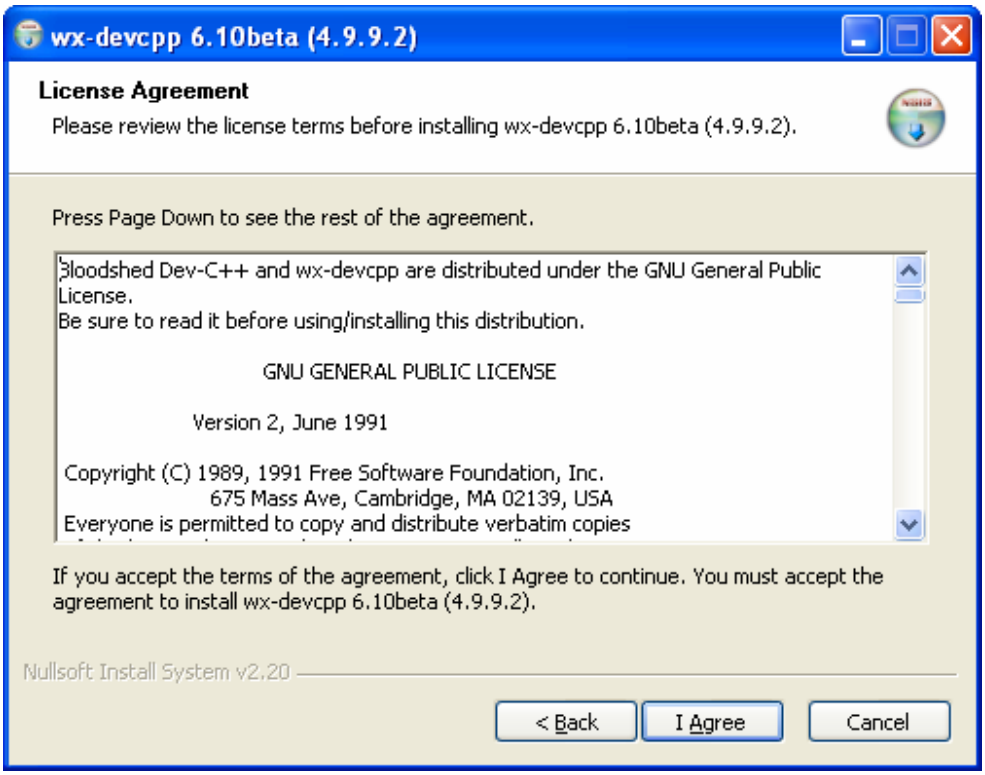


Figure 1.12 – The license agreement dialog

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.

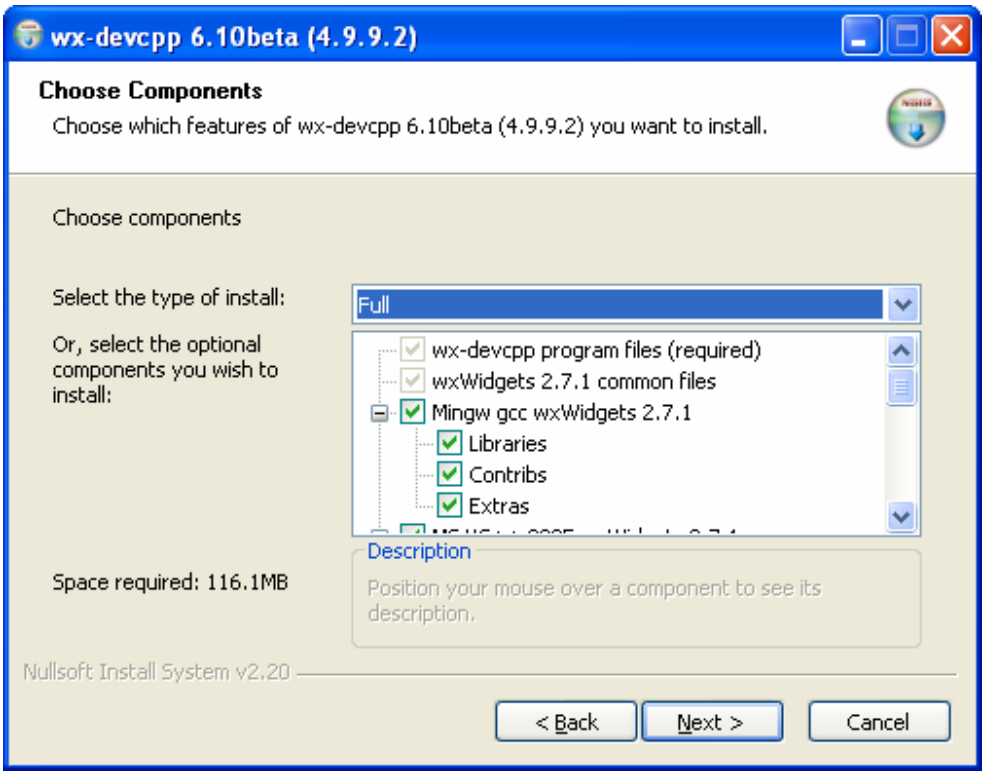


Figure 1.13 – The component choice dialog

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 program together so I change this option to ‘Programming\wxDevCpp’.

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

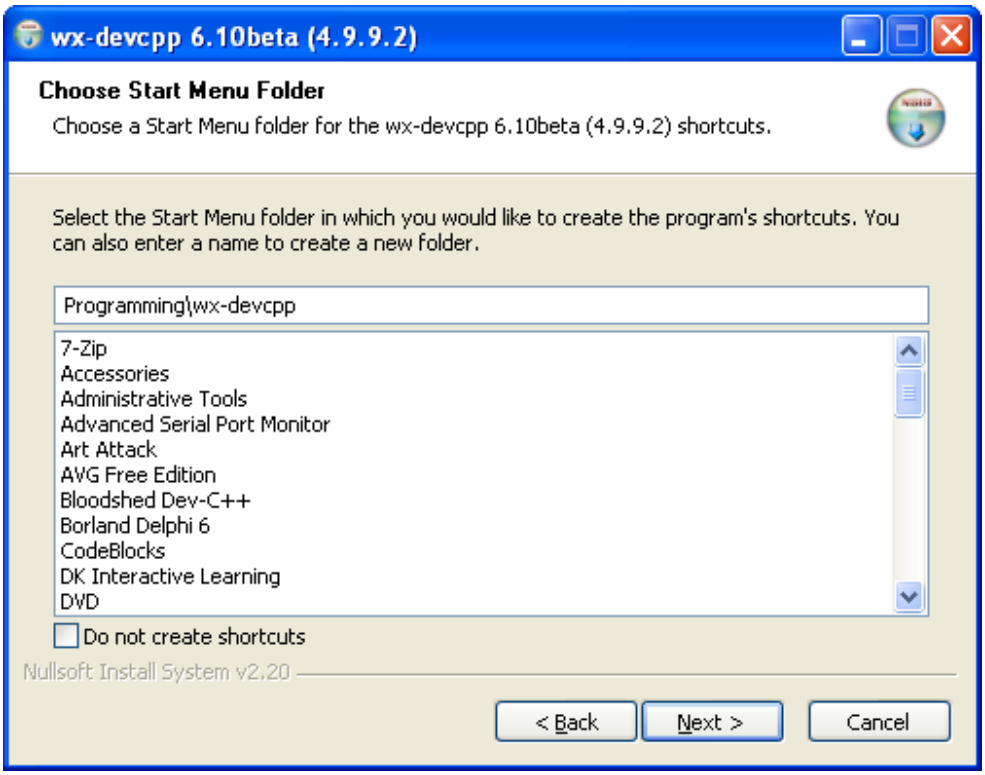


Figure 1.14 – Start menu location option dialog

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].

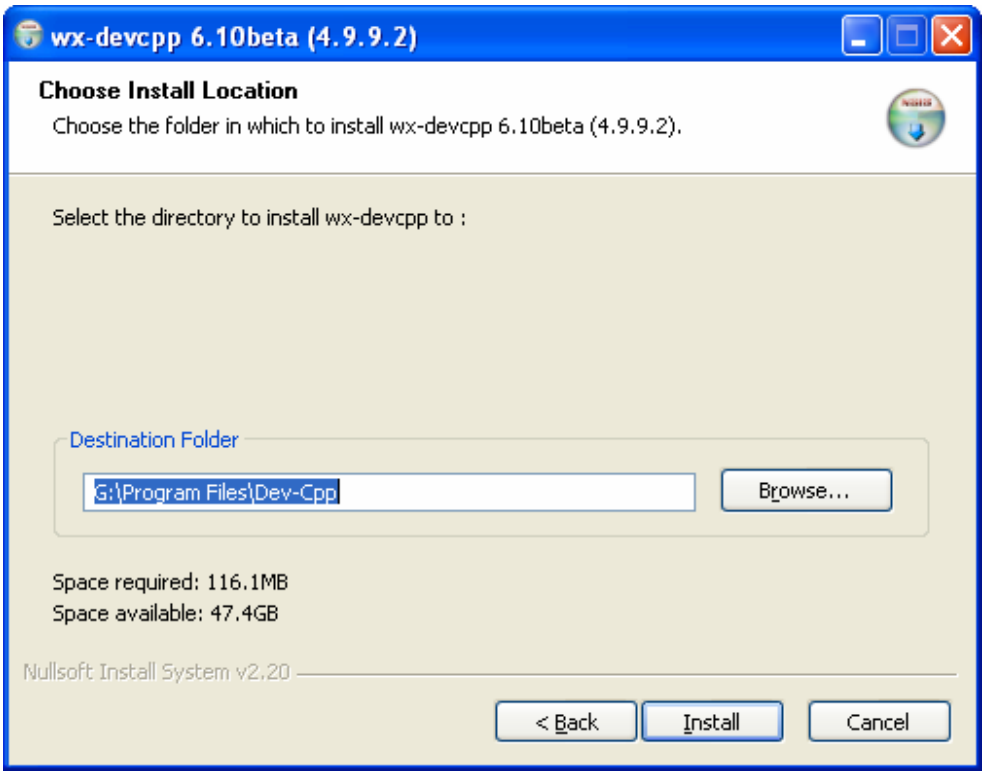


Figure 1.15 – Choose an install location dialog

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

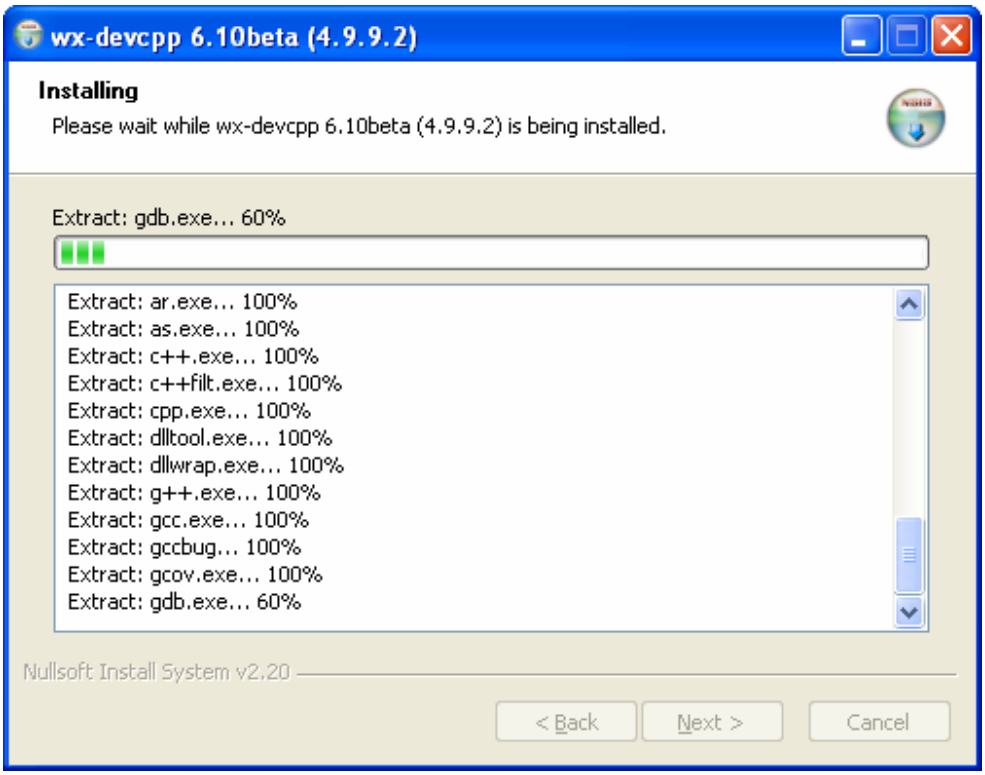


Figure 1.16 – File Installation dialog

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.

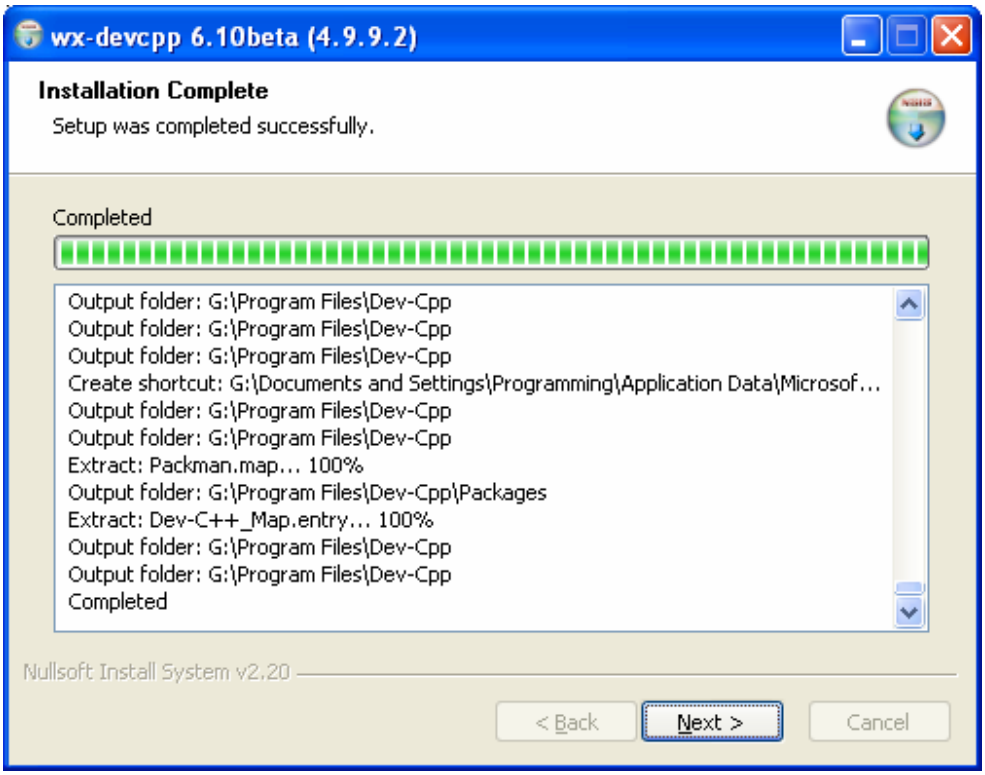


Figure 1.18 – File Installation dialog upon completion

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.

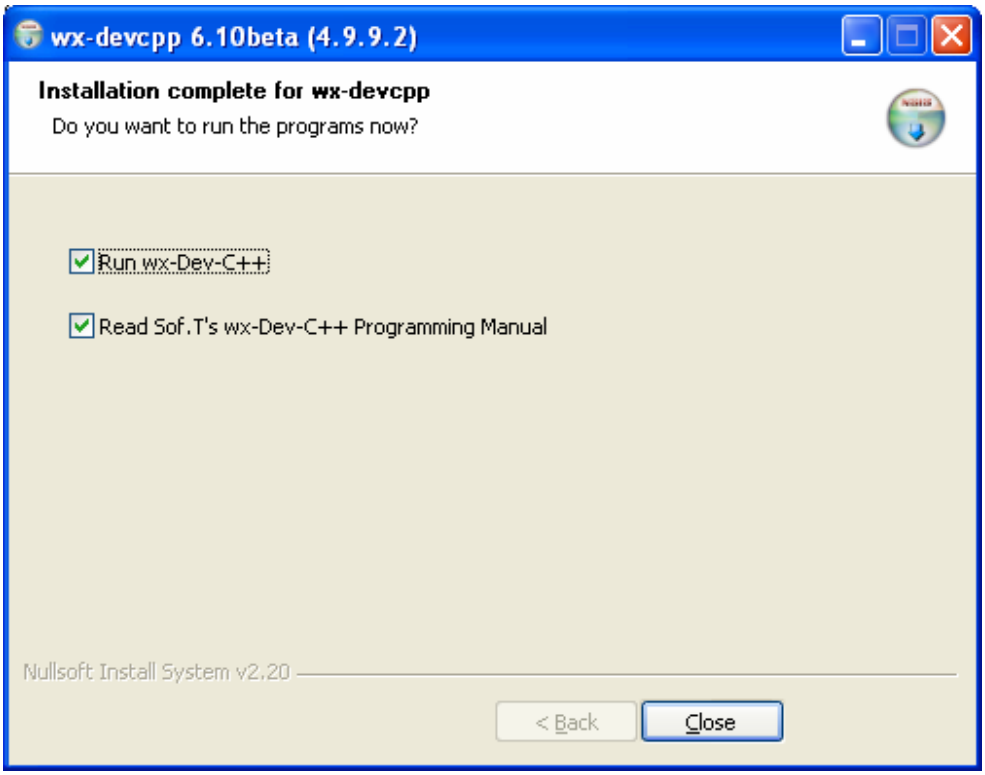


Figure 1.19 – The completion of the setup dialog

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++’.

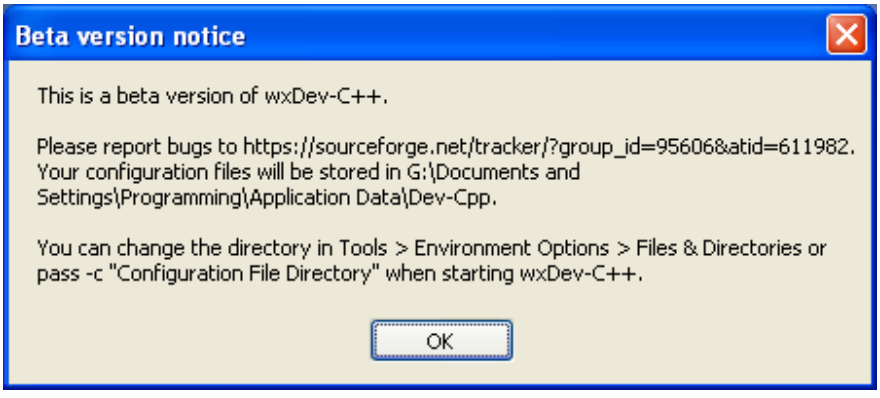


Figure 1.20 – The beta dialog

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.

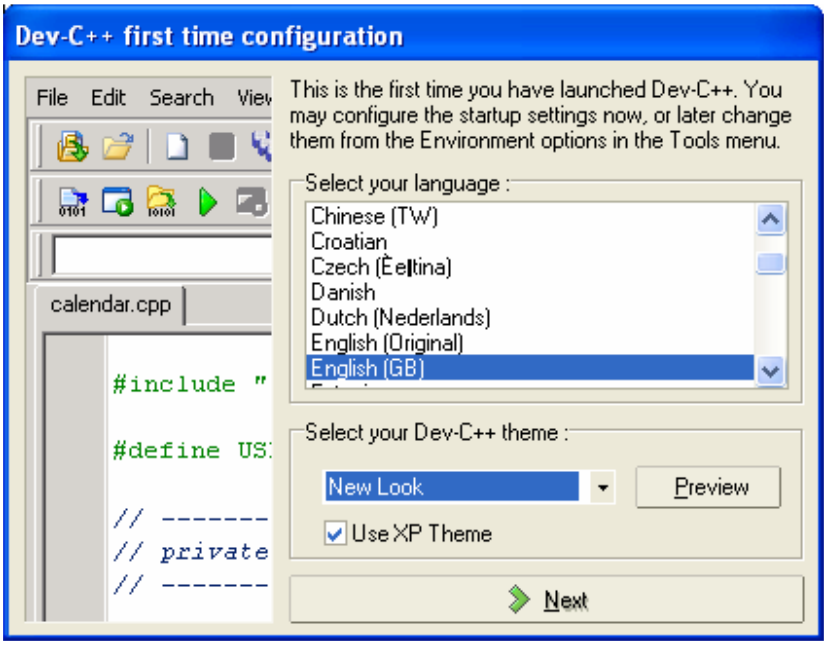


Figure 1.21 – Configuration Dialog

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.

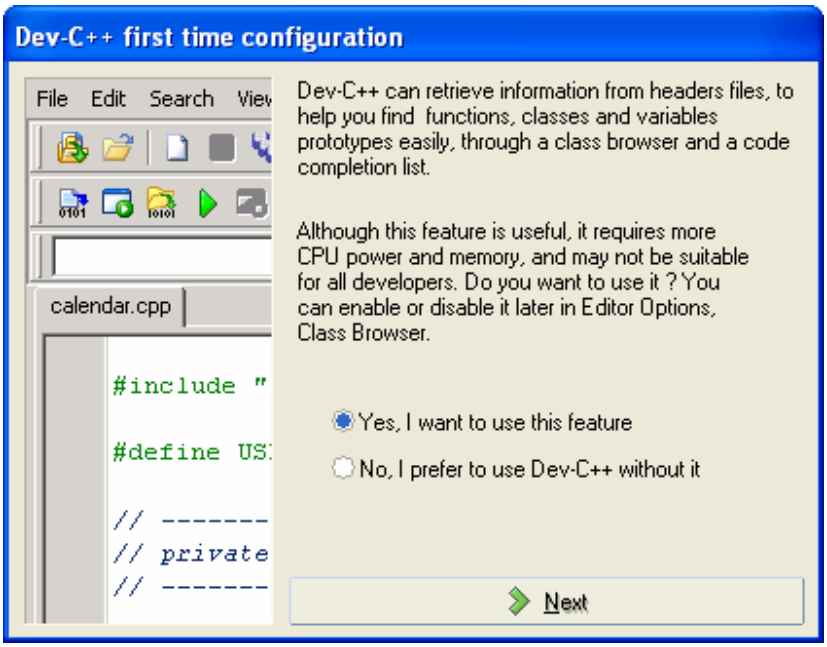


Figure 1.21 – Enable code completion dialog

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.

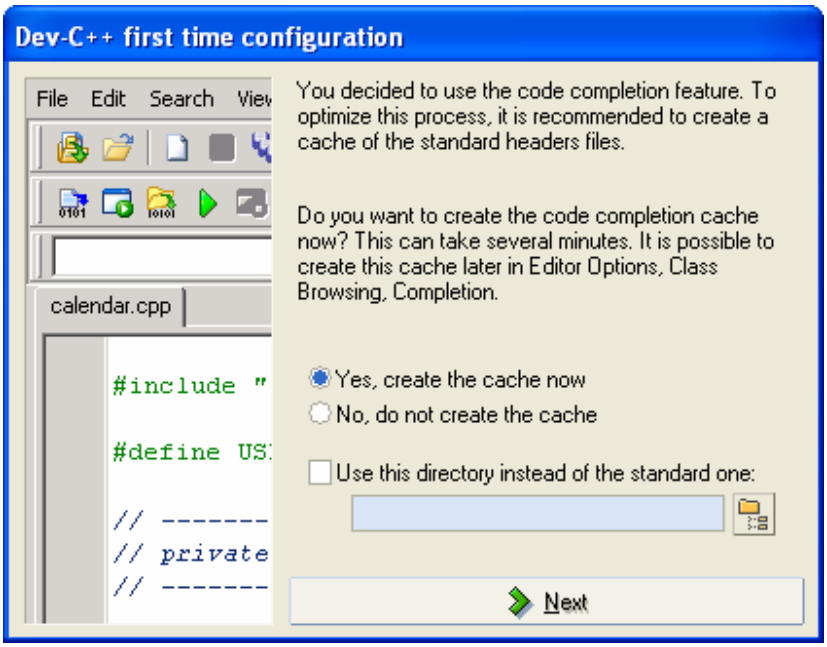


Figure 1.22 – Code completion cache creation dialog (try saying that fast)

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

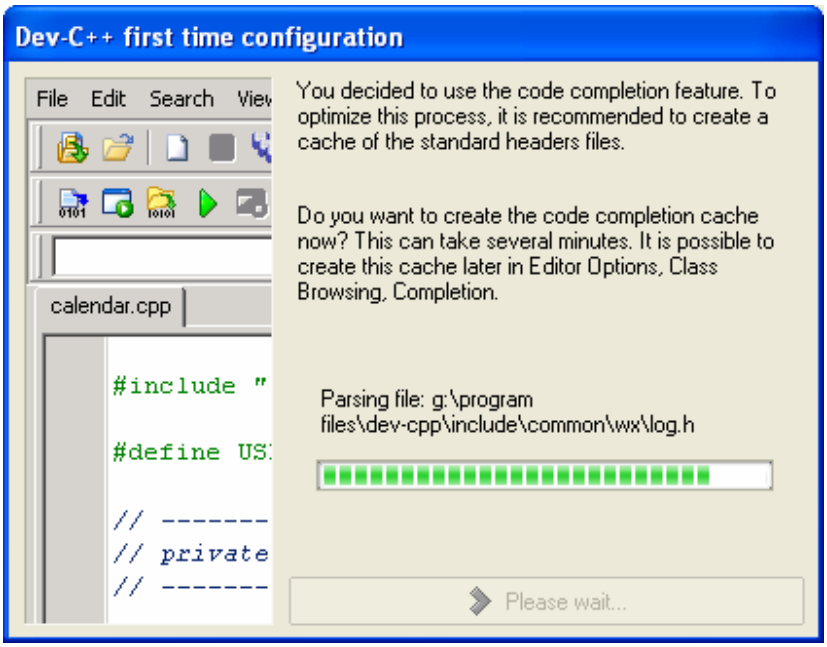


Figure 1.23 – Yep still waiting, nearly finished that coffee though

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

> Click [OK] to complete the installation process.

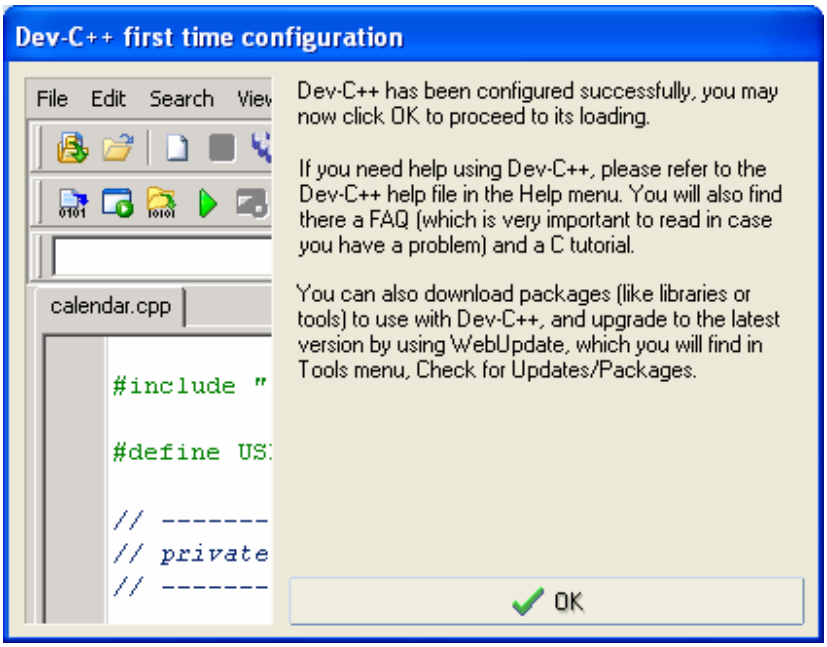


Figure 1.24 – Phew the final dialog

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’).

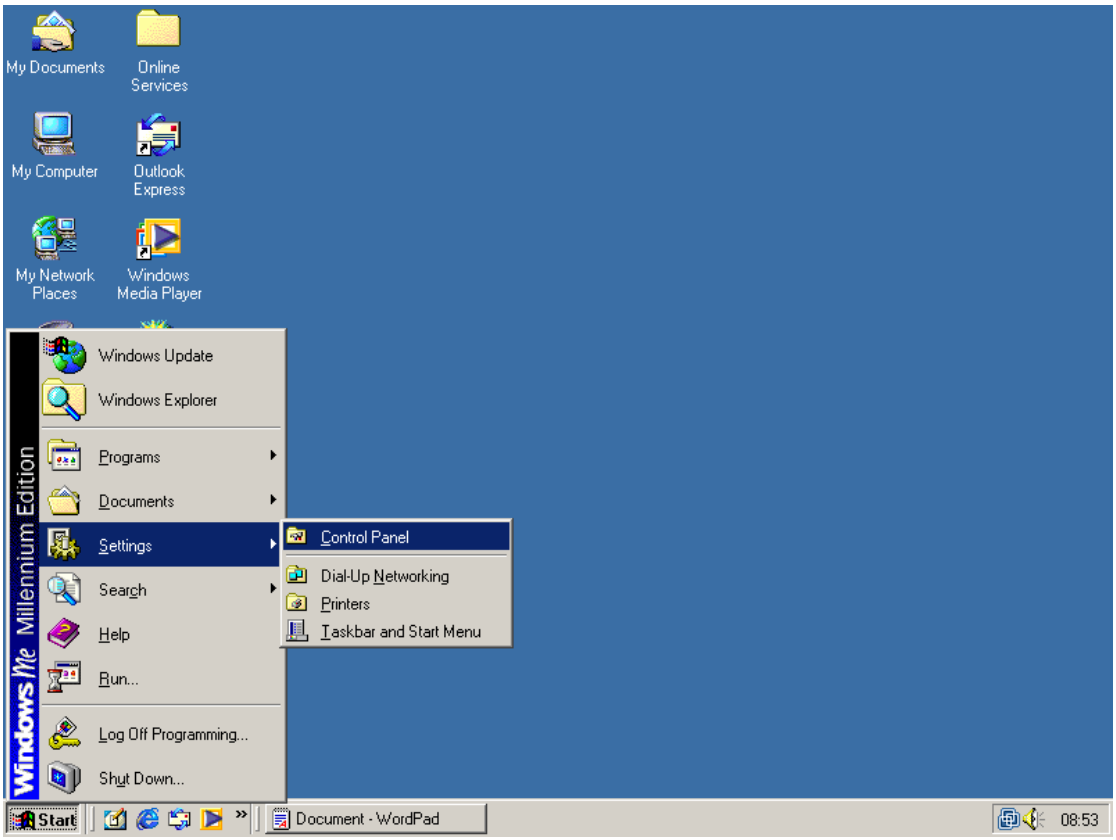


Figure 1.25 – Getting to the Control Panel (Windows 9x)

The Control Panel will pop up with various options.

You need to select Add/Remove Programs.

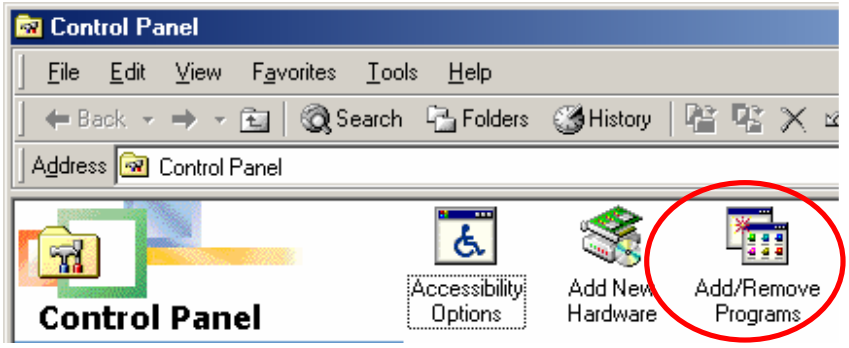


Figure 1.26 – Selecting Add/Remove Programs

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.

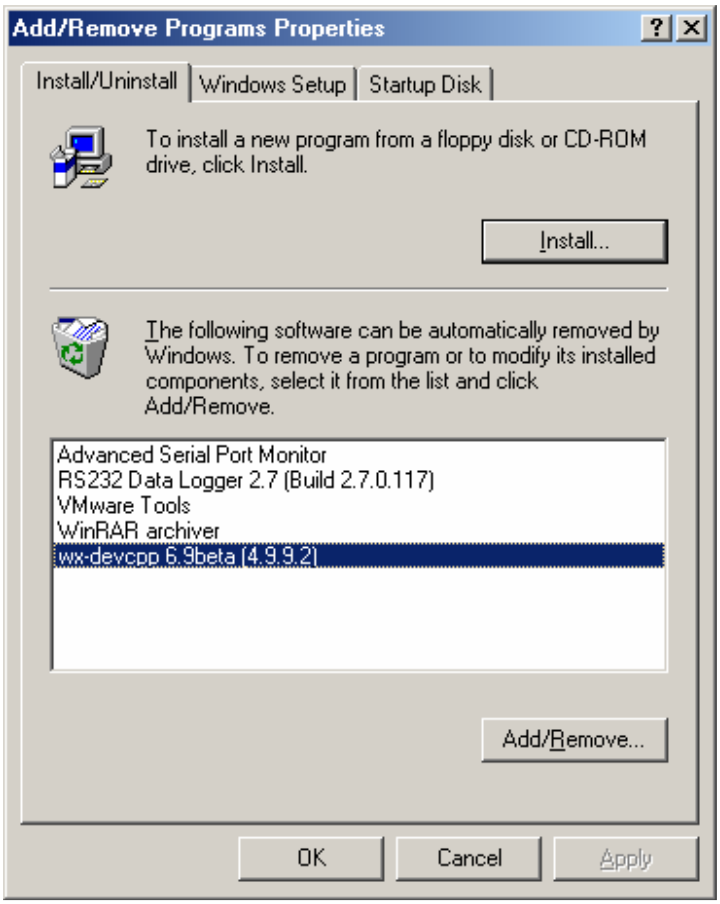


Figure 1.27 – The Add/Remove Dialog

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

Click the [Uninstall] button to continue.

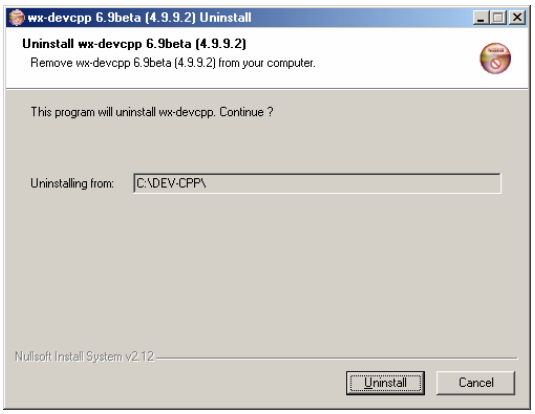


Figure 1.28 – Uninstall dialog

A list of files is displayed as they are uninstalled.

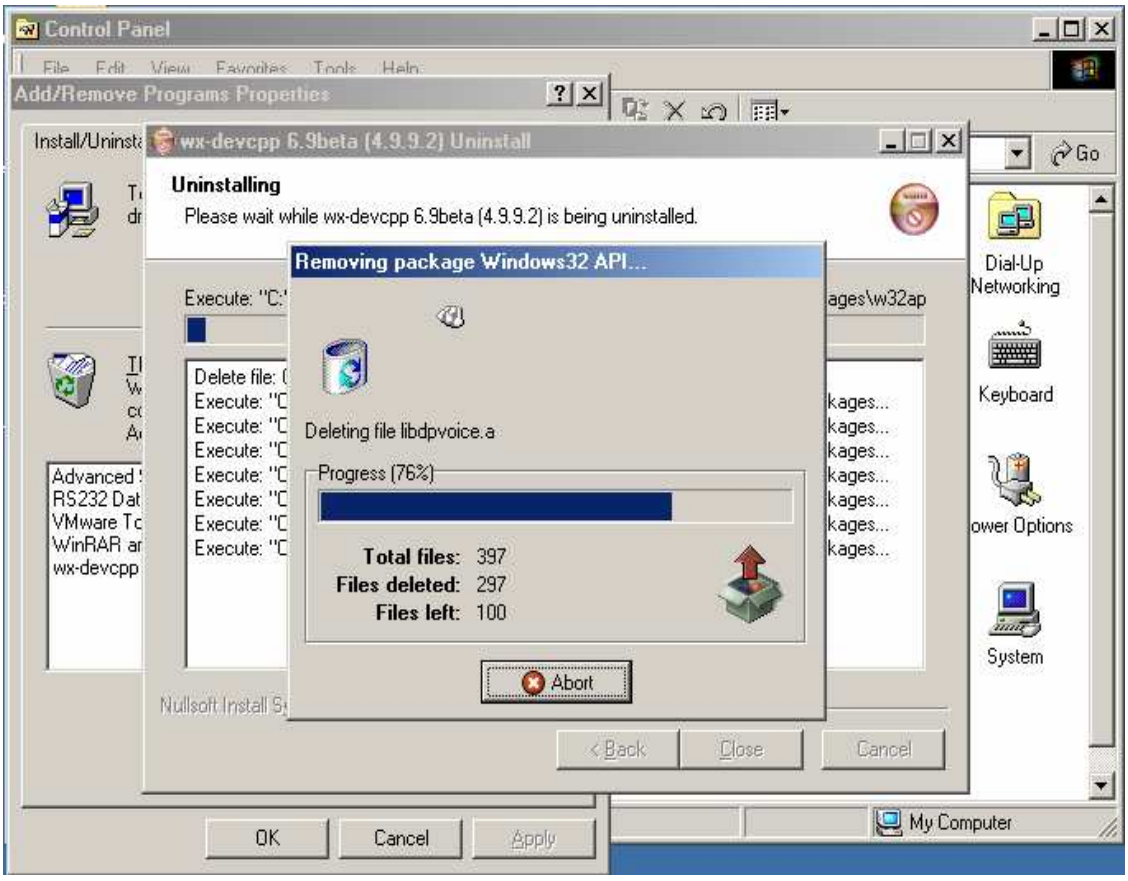


Figure 1.29 – wxDev-C++ being uninstalled.

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

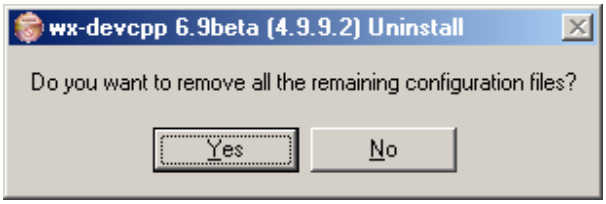


Figure 1.30 – Remove configuration files dialog

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.

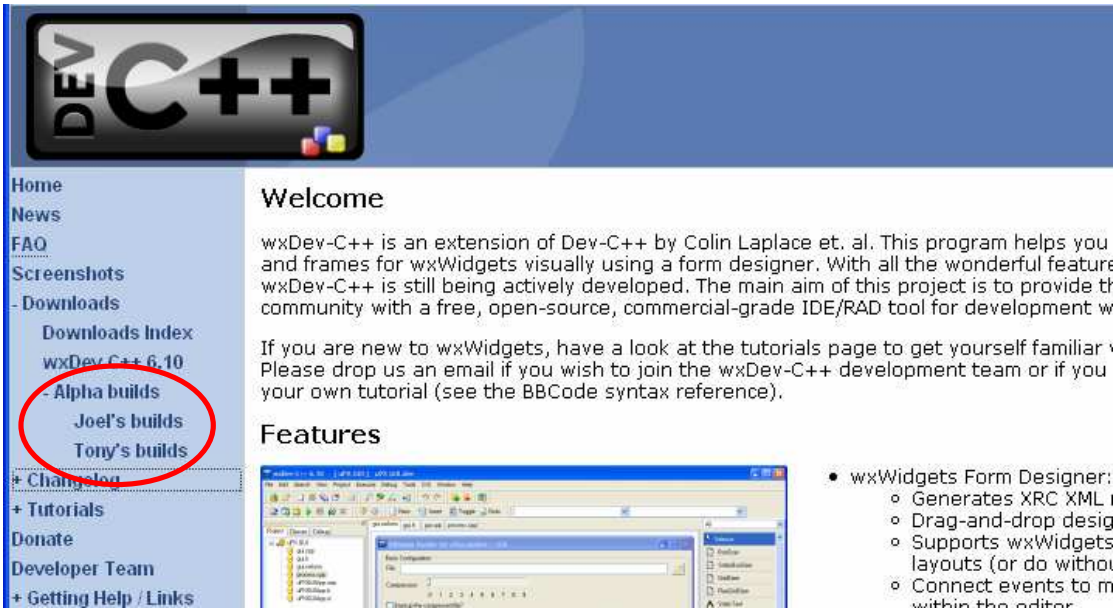


Figure 1.32 – Link to alpha build of the wxDev-C++ IDE

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).

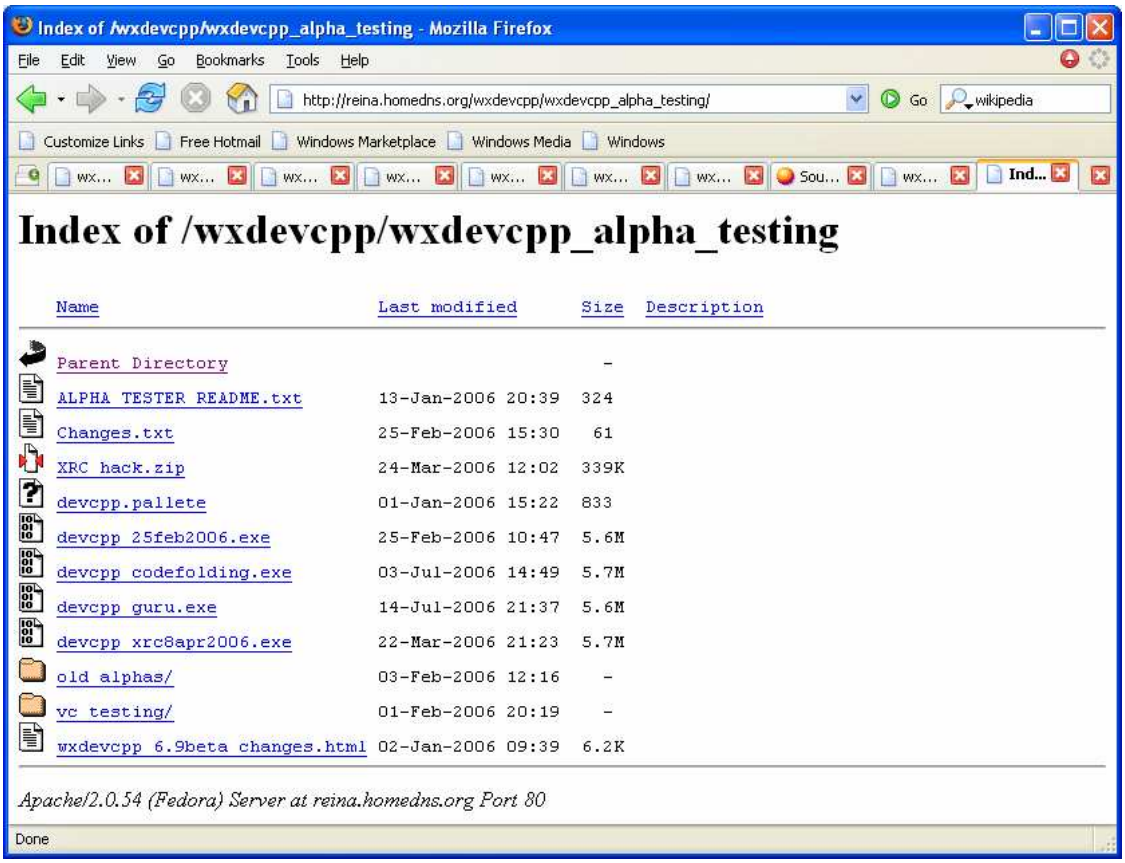
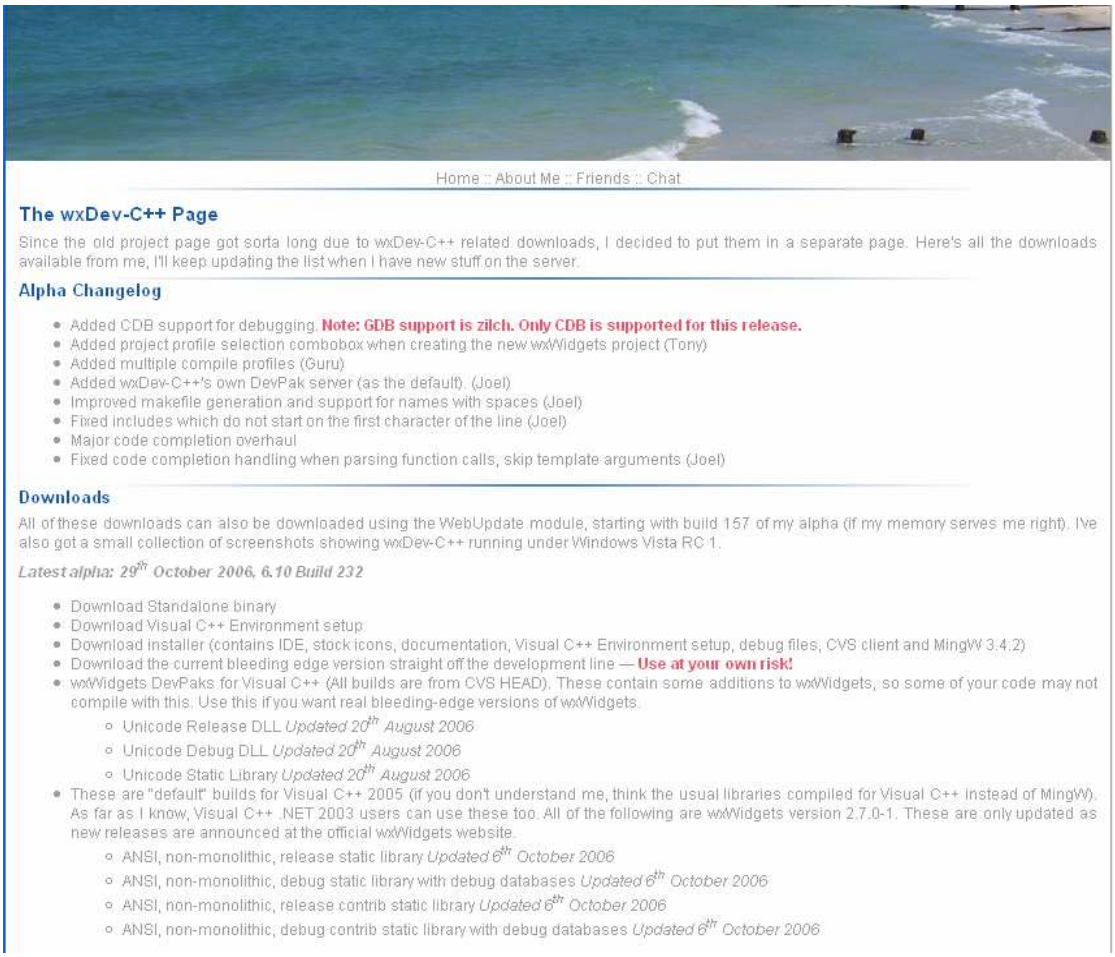


Figure 1.34 – Tony's list of wxDev-C++ versions

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.



Home :: About Me :: Friends :: Chat

The wxDev-C++ Page

Since the old project page got sorta long due to wxDev-C++ related downloads, I decided to put them in a separate page. Here's all the downloads available from me, I'll keep updating the list when I have new stuff on the server.

Alpha Changelog

- Added CDB support for debugging. **Note: GDB support is zilch. Only CDB is supported for this release.**
- Added project profile selection combobox when creating the new wxWidgets project (Tony)
- Added multiple compile profiles (Guru)
- Added wxDev-C++'s own DevPak server (as the default). (Joel)
- Improved makefile generation and support for names with spaces (Joel)
- Fixed includes which do not start on the first character of the line (Joel)
- Major code completion overhaul
- Fixed code completion handling when parsing function calls, skip template arguments (Joel)

Downloads

All of these downloads can also be downloaded using the WebUpdate module, starting with build 157 of my alpha (if my memory serves me right). I've also got a small collection of screenshots showing wxDev-C++ running under Windows Vista RC 1.

Latest alpha: 29th October 2006, 6.10 Build 232

- Download Standalone binary
- Download Visual C++ Environment setup
- Download installer (contains IDE, stock icons, documentation, Visual C++ Environment setup, debug files, CVS client and MingW 3.4.2)
- Download the current bleeding edge version straight off the development line — **Use at your own risk!**
- wxWidgets DevPaks for Visual C++ (All builds are from CVS HEAD). These contain some additions to wxWidgets, so some of your code may not compile with this. Use this if you want real bleeding-edge versions of wxWidgets.
  - Unicode Release DLL *Updated 20th August 2006*
  - Unicode Debug DLL *Updated 20th August 2006*
  - Unicode Static Library *Updated 20th August 2006*
- These are "default" builds for Visual C++ 2005 (if you don't understand me, think the usual libraries compiled for Visual C++ instead of MingW). As far as I know, Visual C++ .NET 2003 users can use these too. All of the following are wxWidgets version 2.7.0-1. These are only updated as new releases are announced at the official wxWidgets website.
  - ANSI, non-monolithic, release static library *Updated 6th October 2006*
  - ANSI, non-monolithic, debug static library with debug databases *Updated 6th October 2006*
  - ANSI, non-monolithic, release contrib static library *Updated 6th October 2006*
  - ANSI, non-monolithic, debug contrib static library with debug databases *Updated 6th October 2006*

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

The latest CVS version should be located at http://home.wanadoo.nl/m.nealon/devcppcvs.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…'

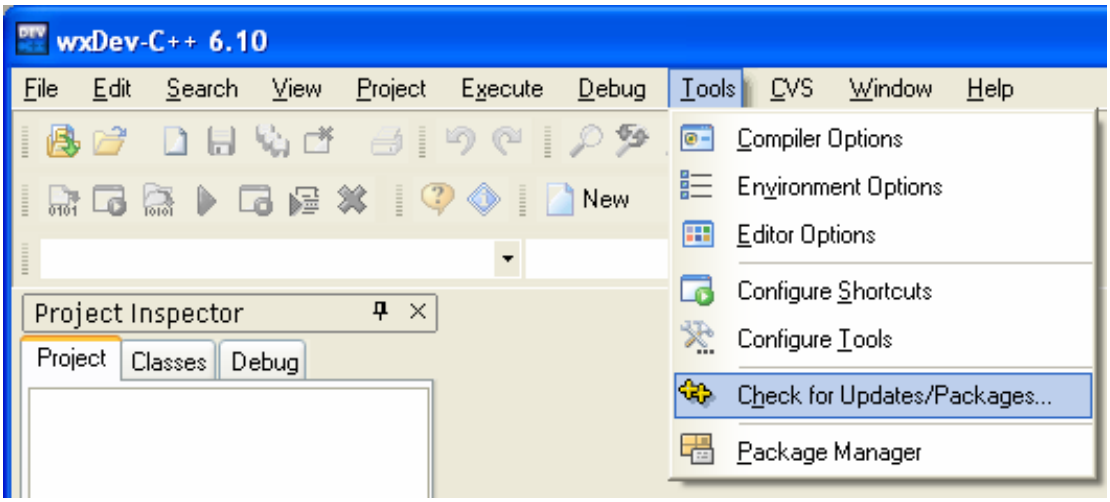


Figure 1.36 – Check for updates from the tool menu

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 labeled ‘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'.

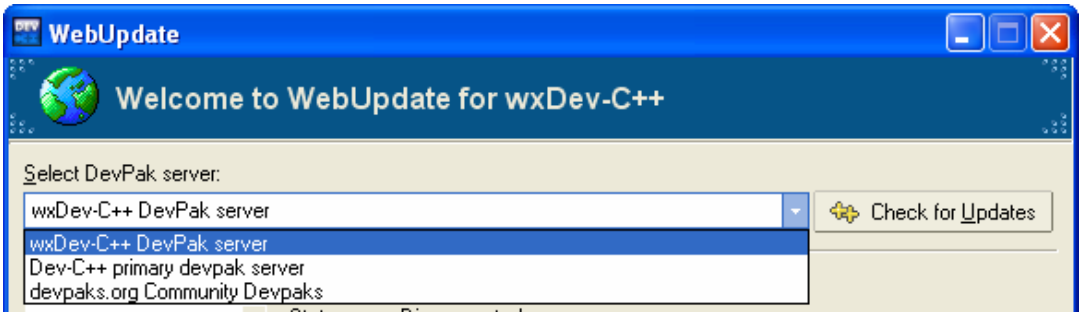


Figure 1.37 – Selecting a DevPak server

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.

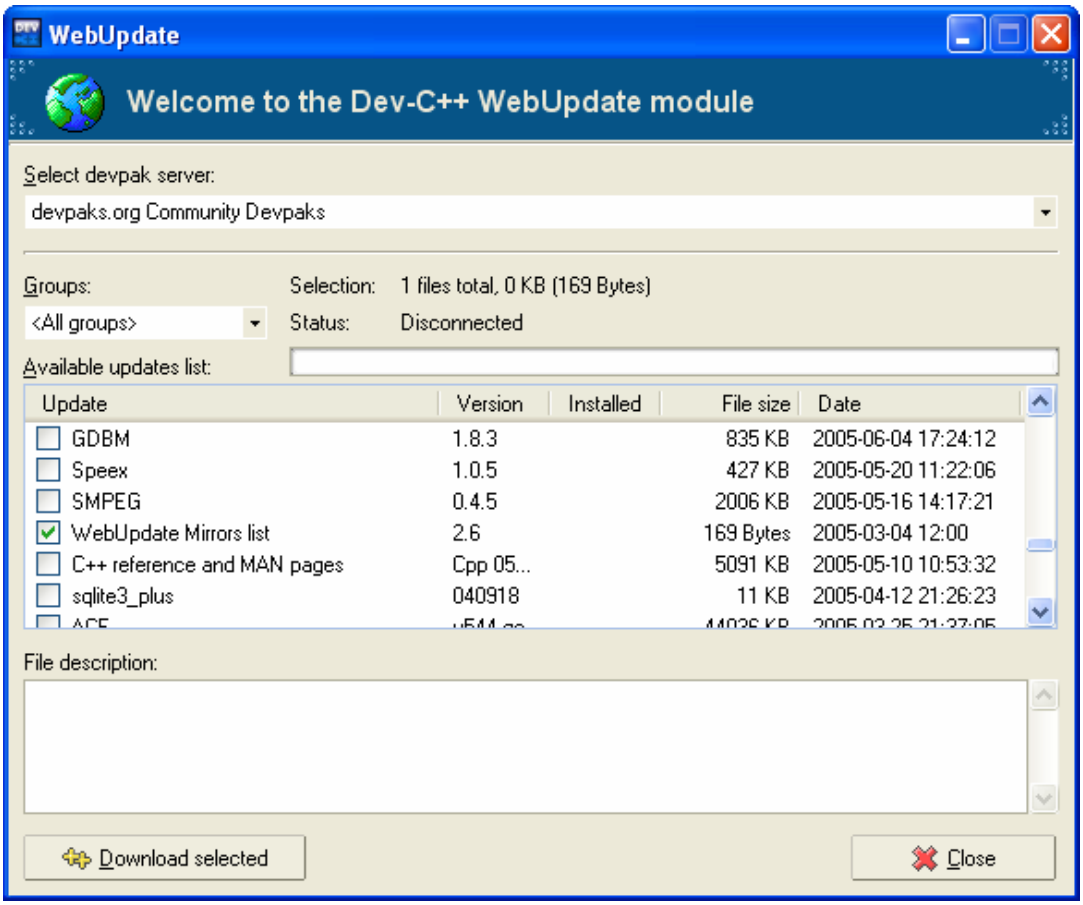


Figure 1.38 – Selecting libraries to download.

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.

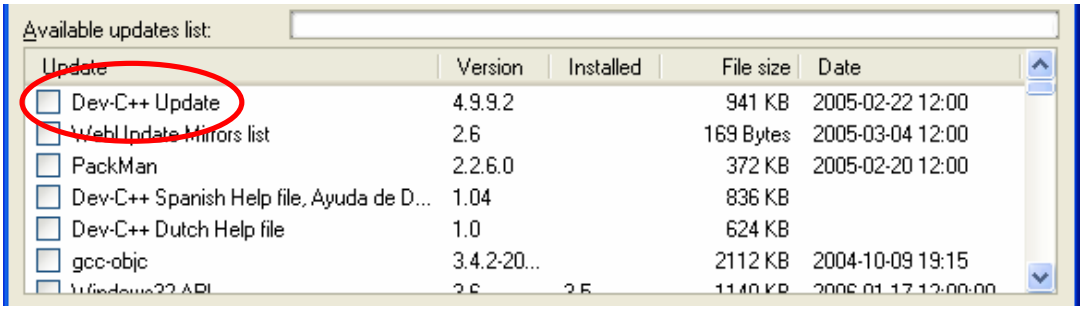


Figure 1.39 – Careful not to download updates of DevC++

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.

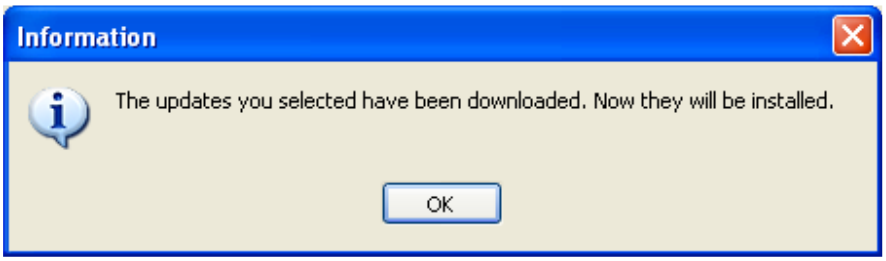


Figure 1.40 – Installing updates dialog

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.

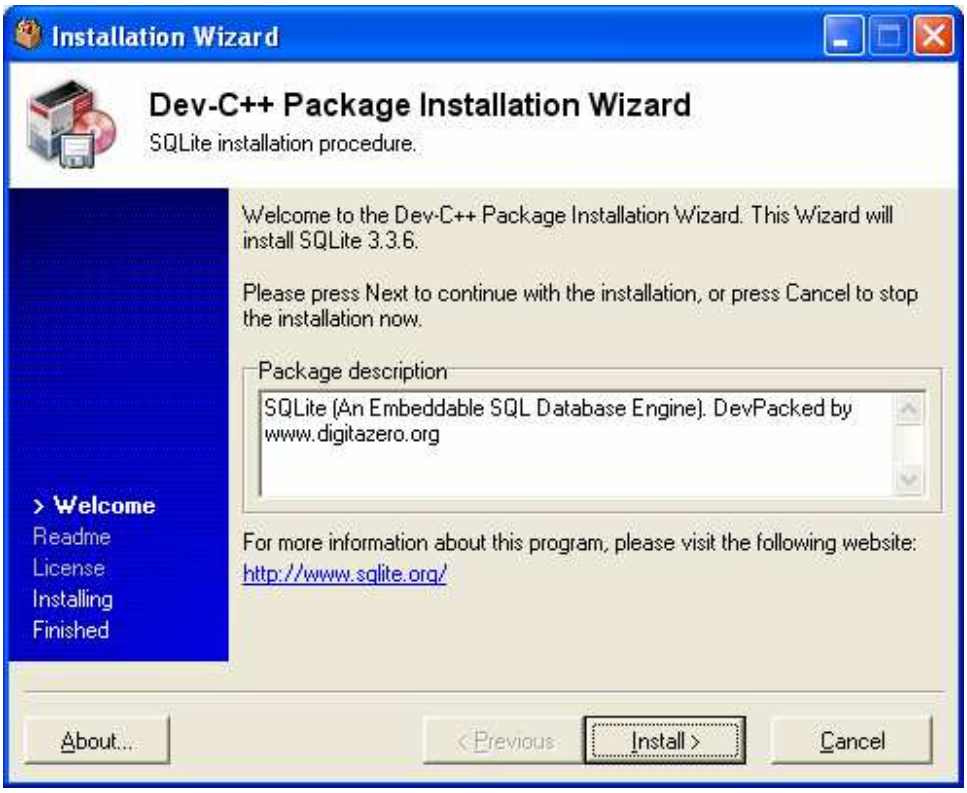


Figure 1.41 – Installing the new package.

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.

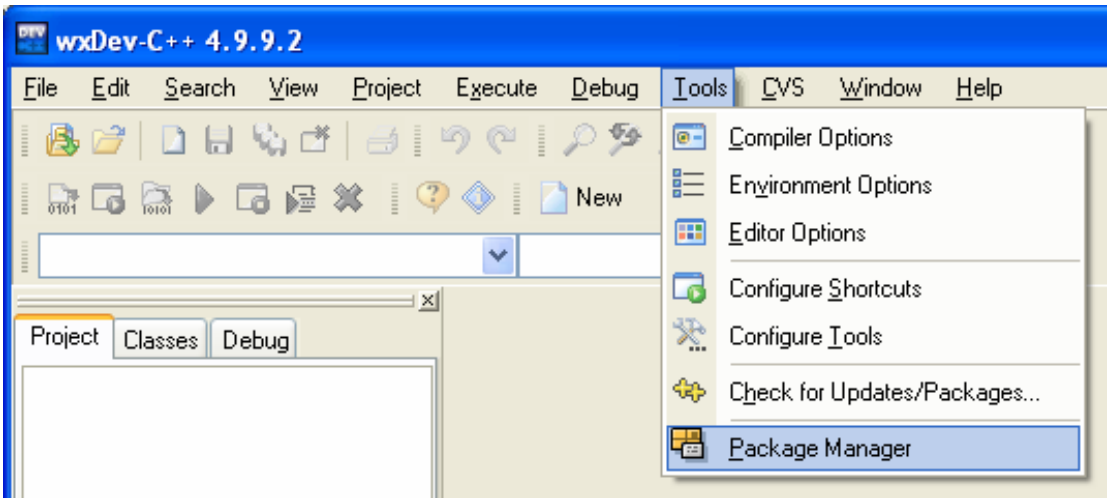


Figure 1.42 – Checking your packages

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.

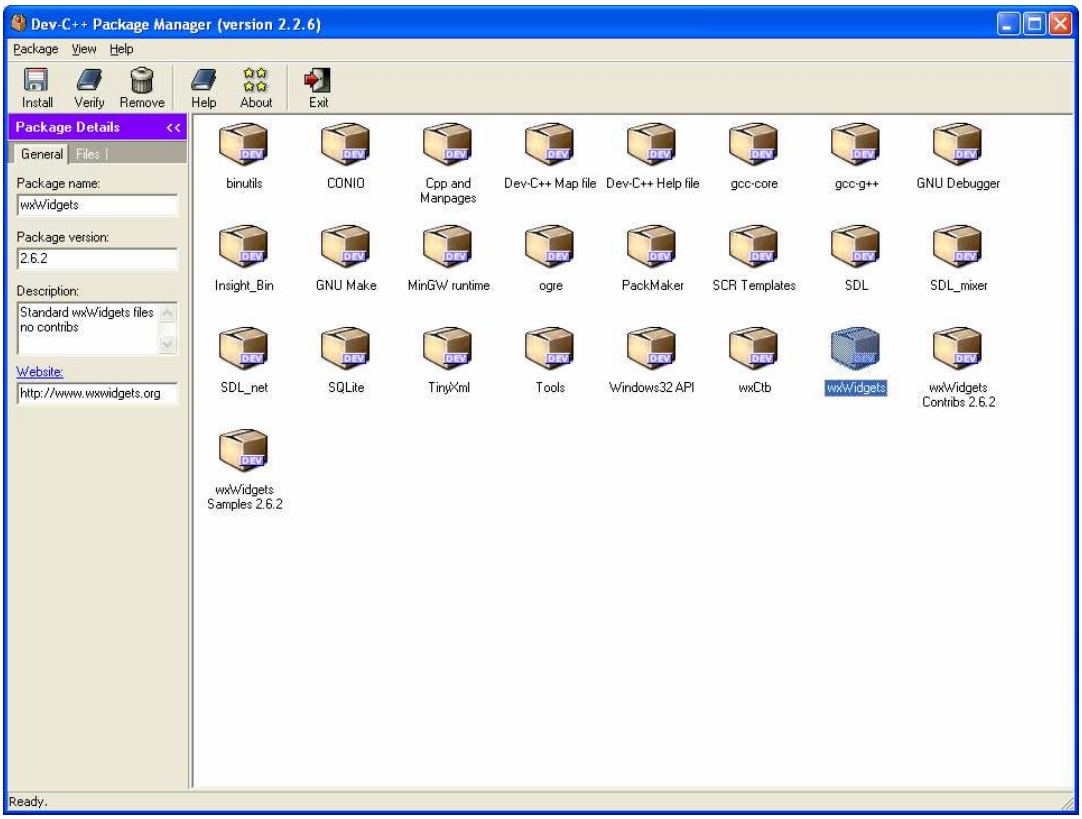


Figure 1.43 – Controlling your packages

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/

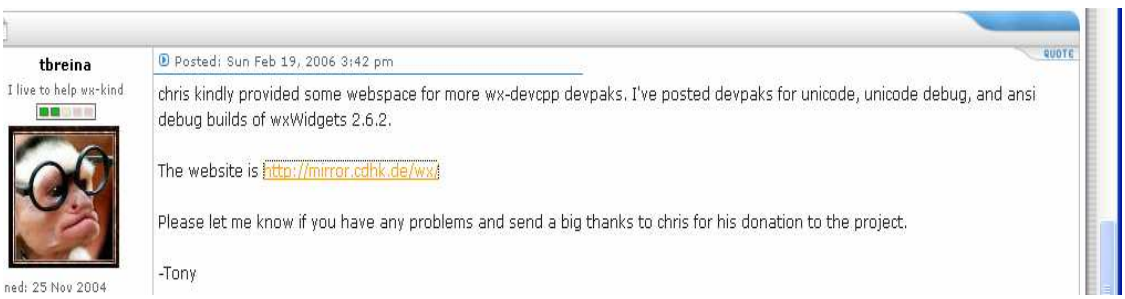


Figure 1.45 – Announcement of a new wxDev-C++ devpak site

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.

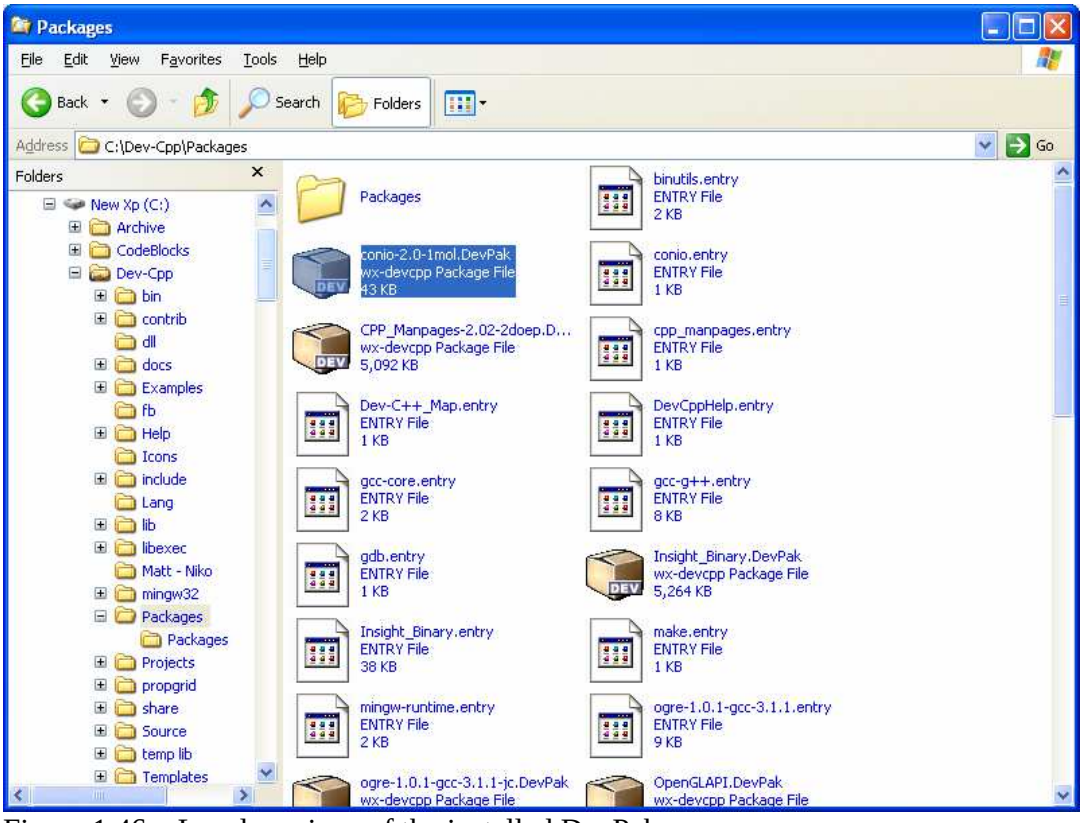


Figure 1.46 – Local versions of the installed DevPaks.

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.

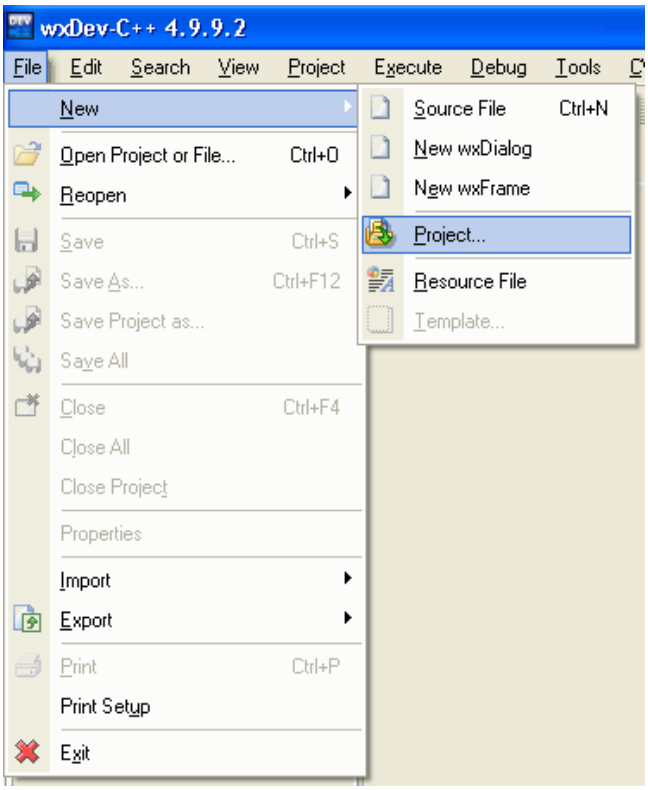


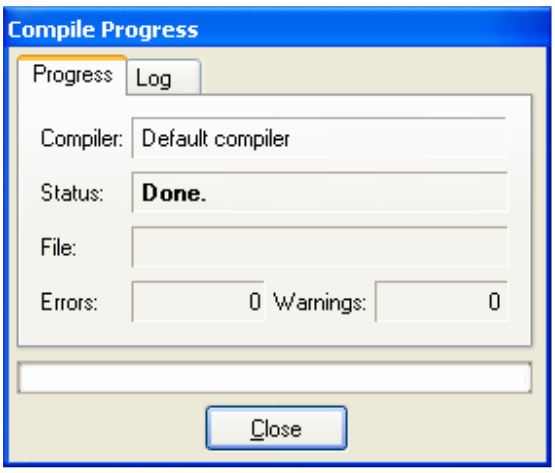
Figure 2.1 – Demonstration of File|New|Project

Keyboard Shortcuts

When you see instructions like press <Ctrl><F9>, this means to hold down the key labeled ‘Ctrl’ on your keyboard and while holding it press the key labeled <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.

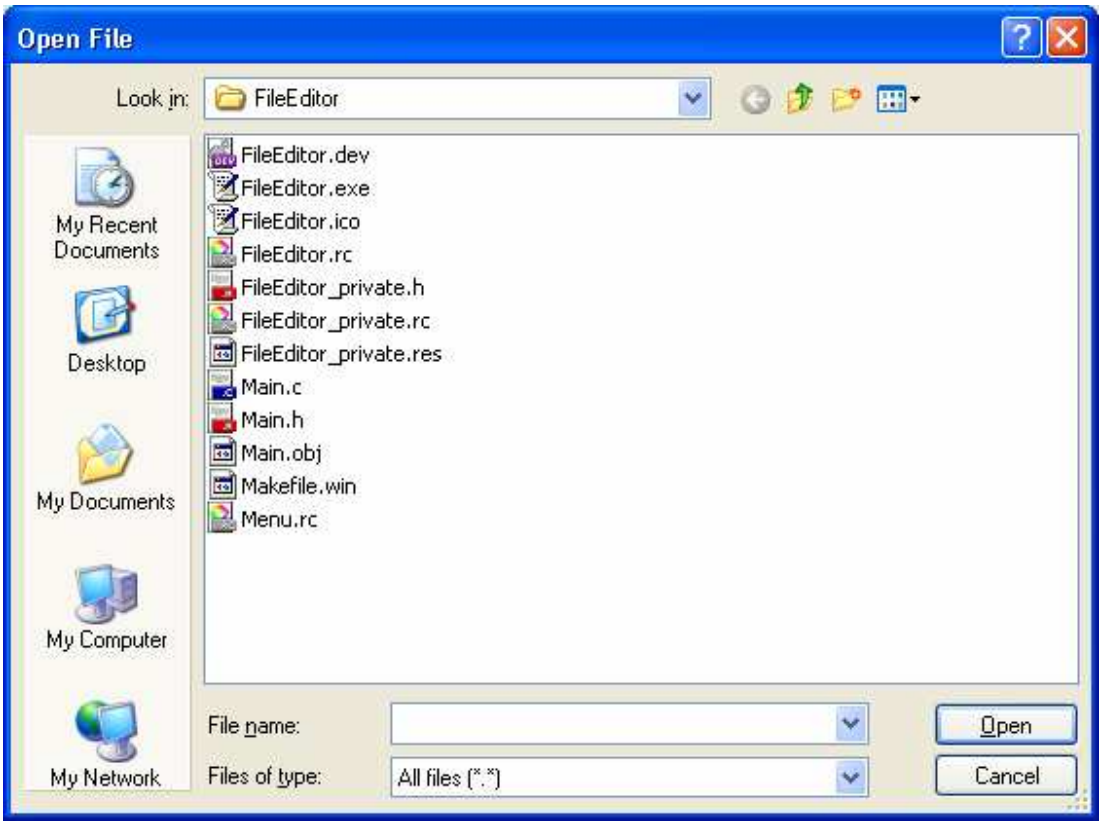


Figure 2.2 – The Open File dialog

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.

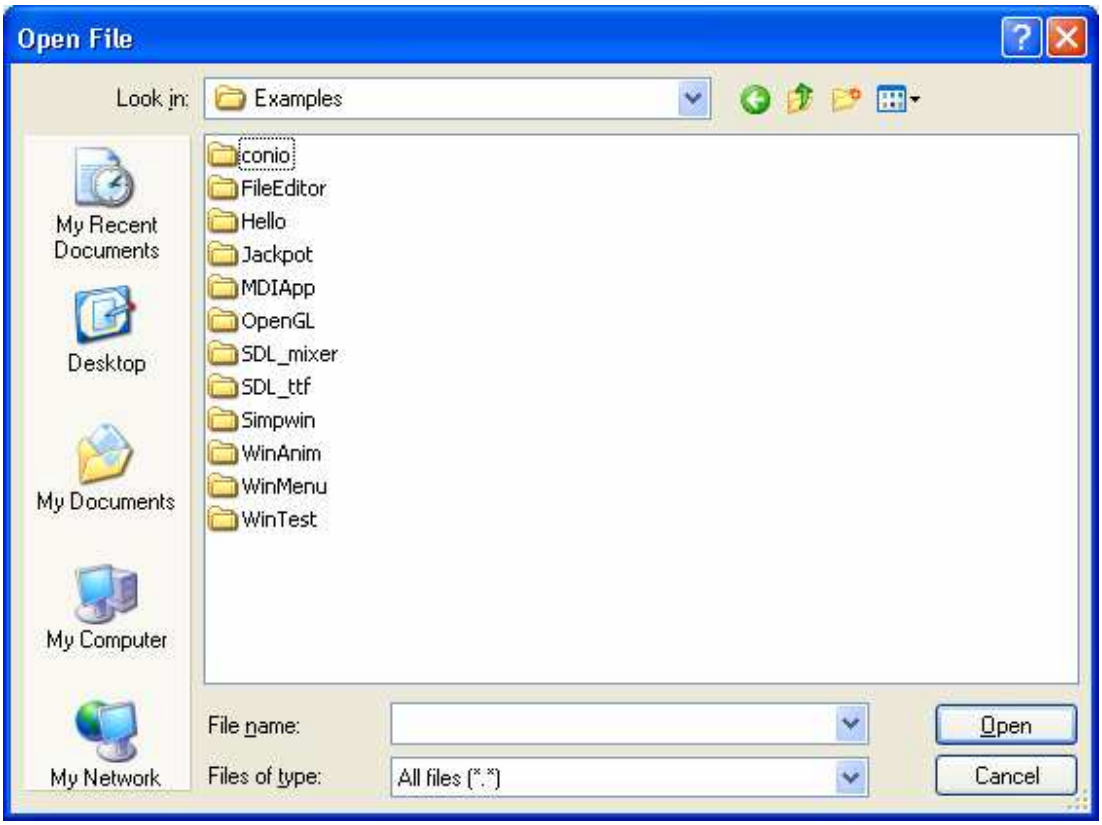


Figure 2.3 – The examples supplied with DevC++/wxDev-C++

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].

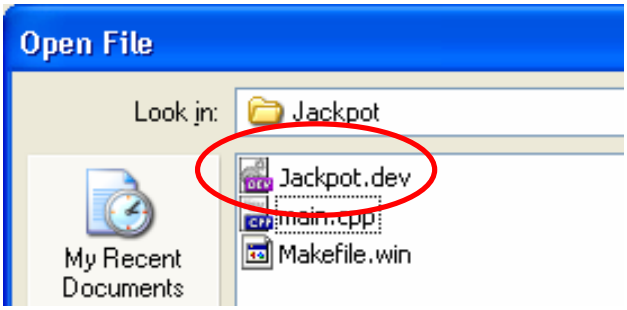


Figure 2.4 – The selecting a .dev project file

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.

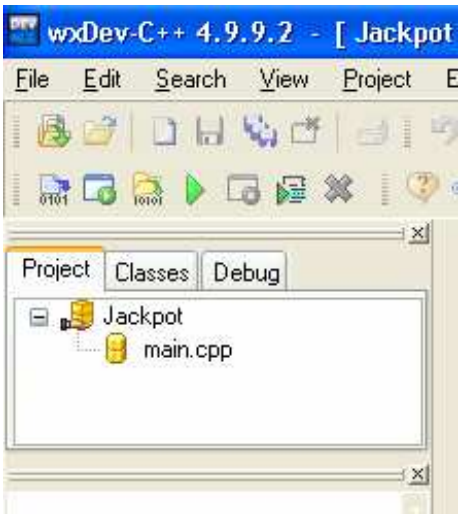


Figure 2.5 – The list of files included in this project

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 are ‘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 difficutly 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 many 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.

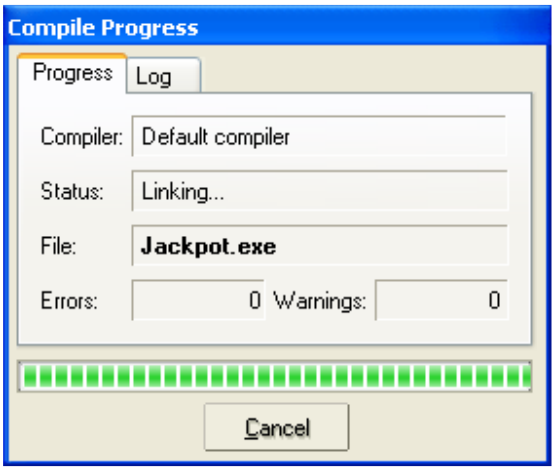


Figure 2.8 – Compiling.

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 with in 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 .

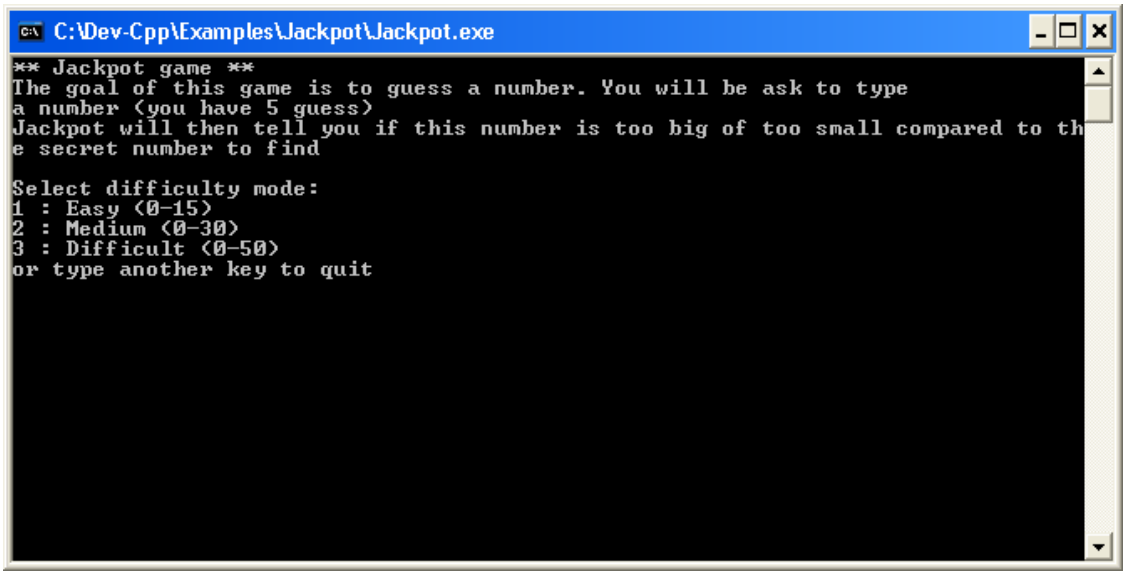


Figure 2.10 – The jackpot game

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.

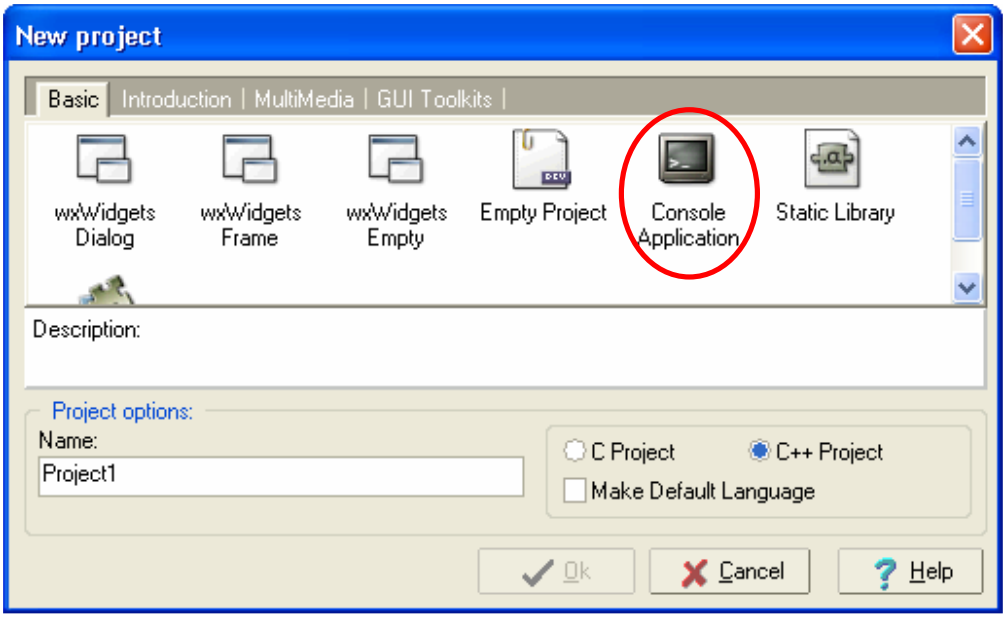


Figure 2.13 – The new project dialog.

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 labeled ‘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.

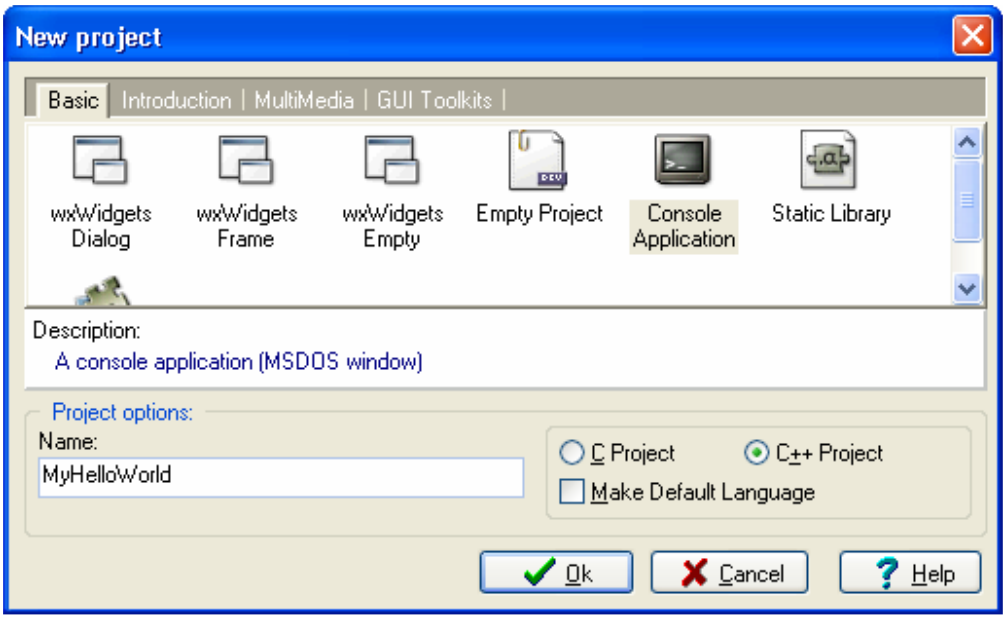


Figure 2.14 – How the New Project dialog should look

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.

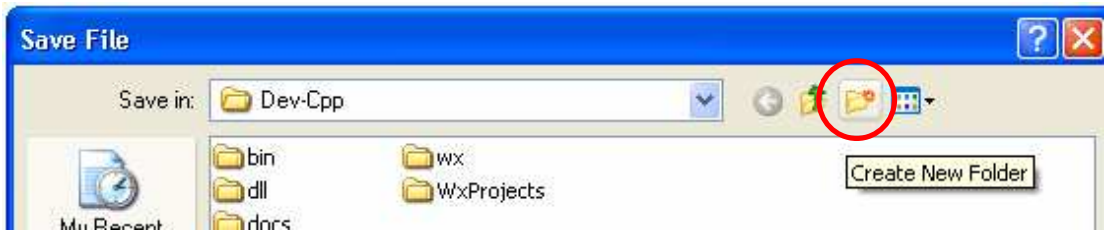


Figure 2.15 – Creating a new folder

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.

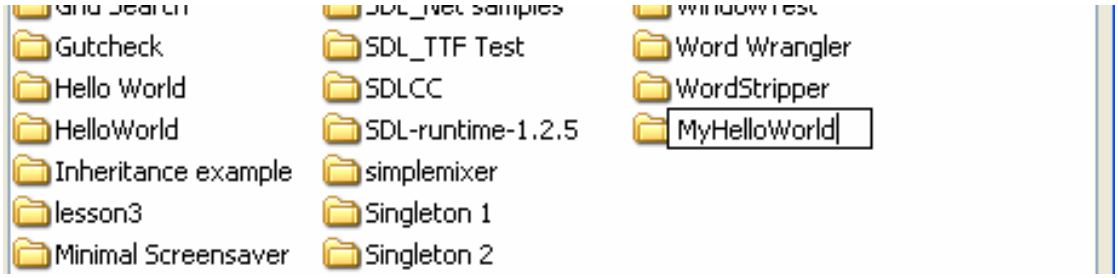


Figure 2.16

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.

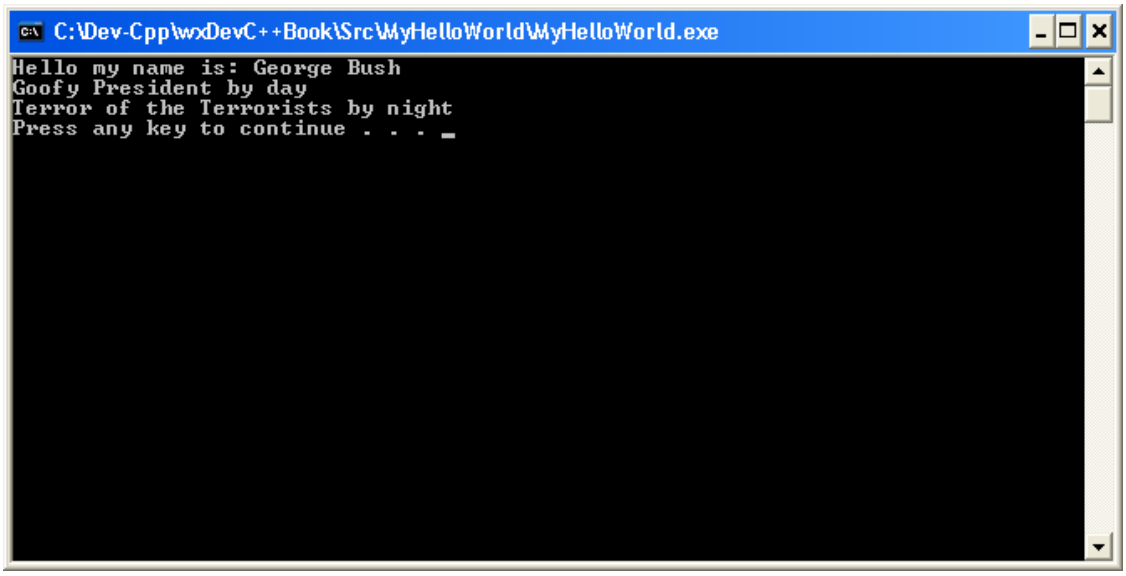


Figure 2.17 – Output from MyHelloWorld program.

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'

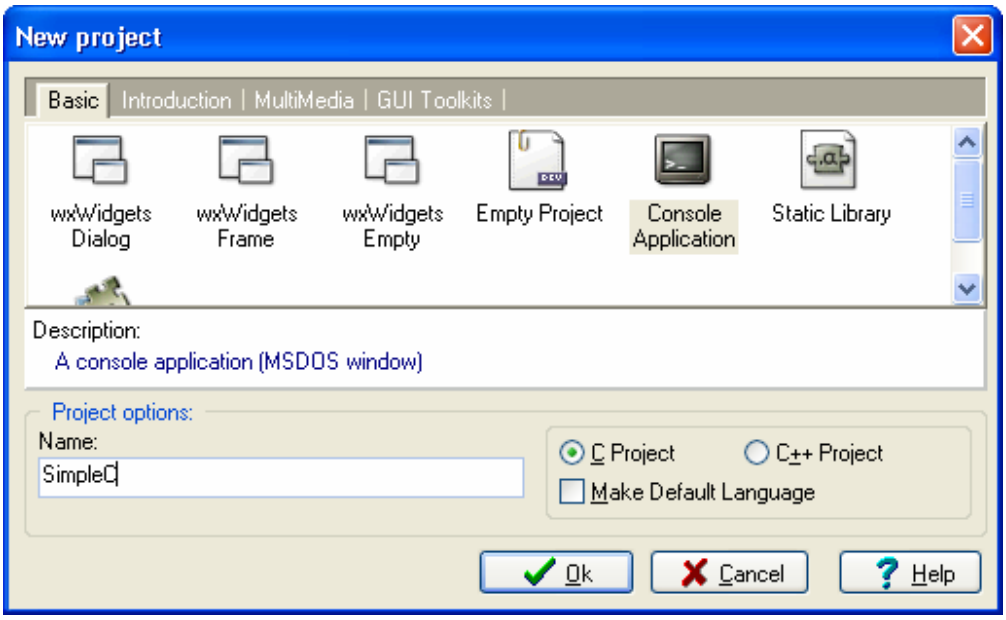


Figure 3.1 – Project settings for the SimpleC program

Click the [OK] button and you will 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 labeled 'Display'
Click on the check box next to 'Line Numbers'

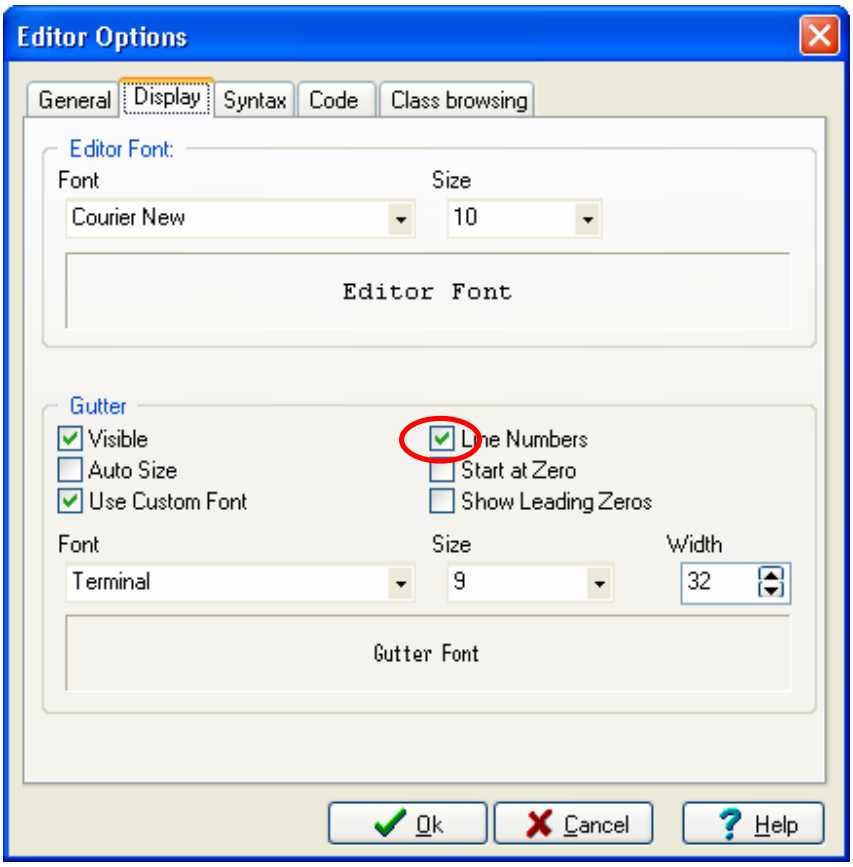


Figure 3.2 – Displaying line numbers

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.

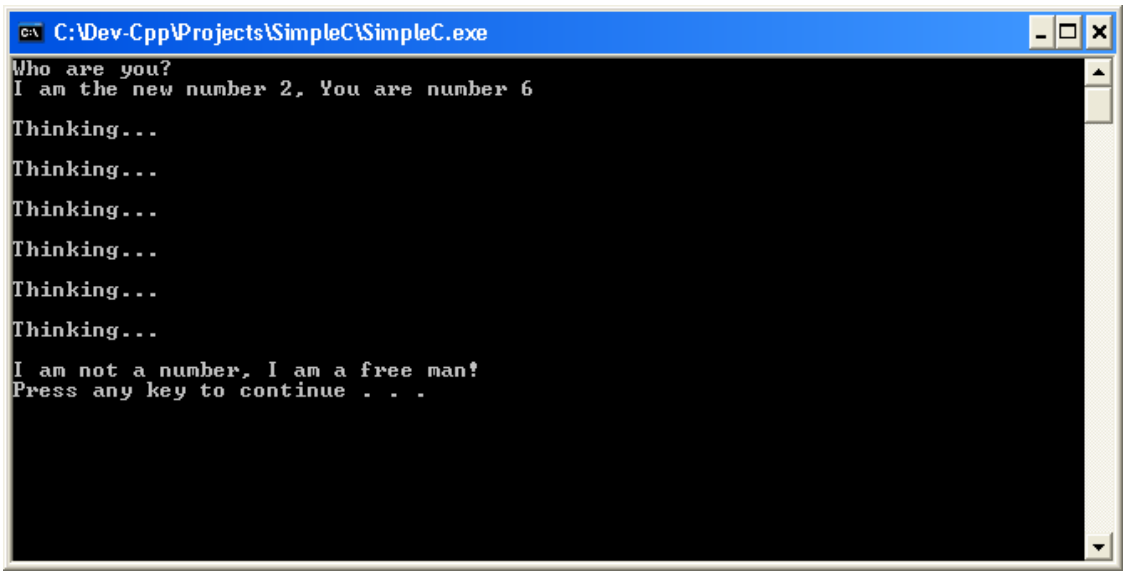


Figure 3.3 – Output from the SimpleC program

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 ‘Prepocessor 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 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.

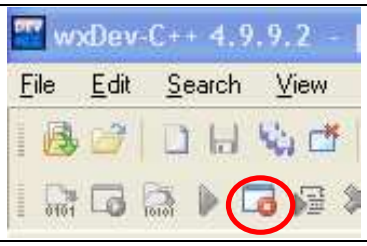
If the `for` loop contains just a single line of code then you can use the shorthand form which leaves out the brackets. For example lines 9 to 12 could have been written as

```
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 a 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 ternery 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 `BobsStatus` 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 ballons\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 ballons\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.

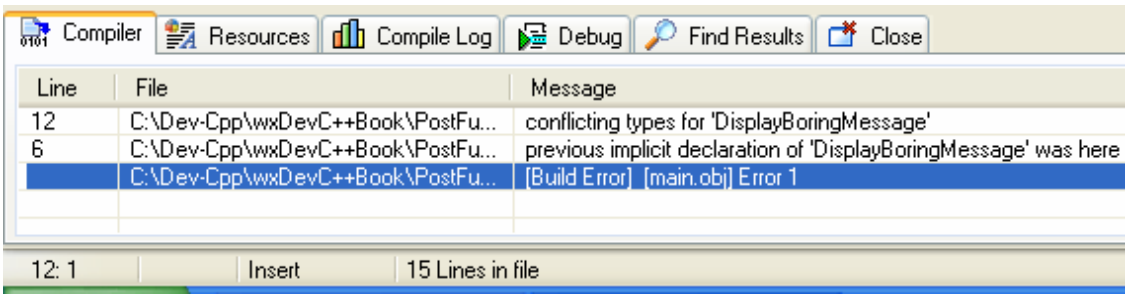


Figure 3.4 – Error message.

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	Ouput 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	Ouput 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 specifiy 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.

NOTE:	Remember when we discussed variable argument functions we said that `va_arg` needed to be told what data type was expected. That is the reason for the different format specifiers. Each time `printf` 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 `printf` 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.

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 specifers, 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 %*.*f\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

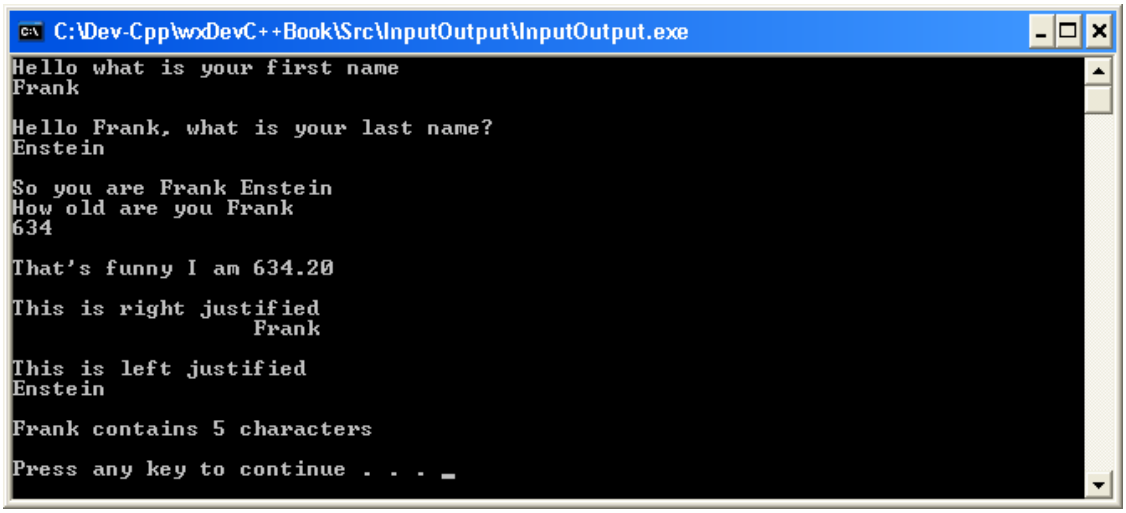


Figure 3.5 – Output from InputOutput program

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

Line 18 where the %*.*f 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>

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

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 = *MyIntPointer;
```

This assigns the value referenced by MyIntPointer 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 * MyIntPointer = &AnInteger;
```

Otherwise you can assign it the value `NULL`.

```
int * MyIntPointer = 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 * MyIntPointer;
8
9         /*Use MyIntPointer to reference Total. We need to use '&'*/
10        MyIntPointer = &Total;
11        /*Use MyIntPointer to display value and location of variable Total*/
12        printf("The value of Total =  %d.\n", *MyIntPointer);
13        printf("The Location of Total is %d\n", MyIntPointer);
14        /*Use MyIntPointer to alter the value of Total*/
15        *MyIntPointer = Number1 * Number2;
16        /*Display the location and value of Total
17        displaying that MyIntPointer 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 MyIntPointer.

Line 12 - uses the ‘*’ operator to print out the value held in the memory location pointer to by MyIntPointer.

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

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

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

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

First we declare MyIntPointer 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 MyIntPointer. Finally we check the MyIntPointer does not point to `NULL` before using it.

`calloc` The memory returned by `malloc` contains random values. `calloc` intializes 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 * MyIntPointer = NULL;
MyIntPointer = (int*) calloc(1,sizeof(int));
If(MyIntPointer)
...
```

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 protype 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.

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

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

```
prinf( "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 to with them. We can compare strings, add strings to other strings, find out how long they are and much more.

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

String functions – Contained in <string.h>

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

Character functions – Contained in <ctype.h>

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

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

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

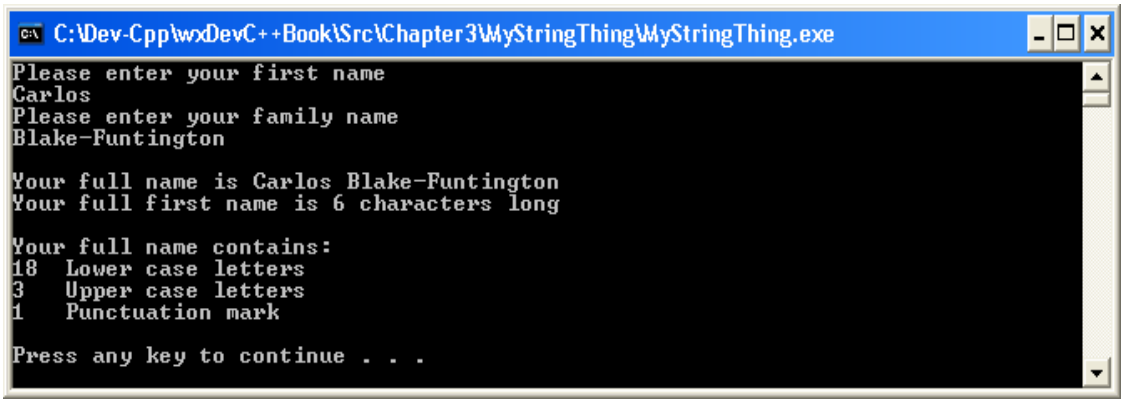


Figure 3.6 – Output from the MyStringThing Program

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 contains 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 of 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 OldMacsFarm;
22        /*Create a union*/
23        FarmYardUnion YoungMacsFarm;
24
25        /*Fill the members of OldMacdonaldsFarm structure*/
26        OldMacsFarm.NumberOfCows = 12;
27        OldMacsFarm.NumberOfSheep = 54;
28        OldMacsFarm.NumberOfPigs = 6;
29
30        /*Fill the members of YoungMacdonaldsFarm union*/
31        YoungMacsFarm.NumberOfCows = 12;
32        YoungMacsFarm.NumberOfSheep = 54;
33        YoungMacsFarm.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",OldMacsFarm.NumberOfCows);
38        printf("and on that farm he had %d sheep\n",OldMacsFarm.NumberOfSheep);
39        printf("and on that farm he had %d pigs\n\n",OldMacsFarm.NumberOfPigs);
40
41        printf("Young Macdonald also had a farm\n");
42        printf("On that farm he had %d cows\n",YoungMacsFarm.NumberOfCows);
43        printf("and on that farm he had %d
          sheep\n",YoungMacsFarm.NumberOfSheep);
44        printf("and on that farm he had %d
          pigs\n\n",YoungMacsFarm.NumberOfPigs);
45
46        /*Print out the sizes of the structure compared to the union*/
47        printf("Sizeof Farmyard struct is %d\n",sizeof(FarmYardStruct));
48        printf("Sizeof Farmyard union is %d\n",sizeof(FarmYardUnion));
49
50        system("PAUSE");
51        return 0;
52      }
```

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

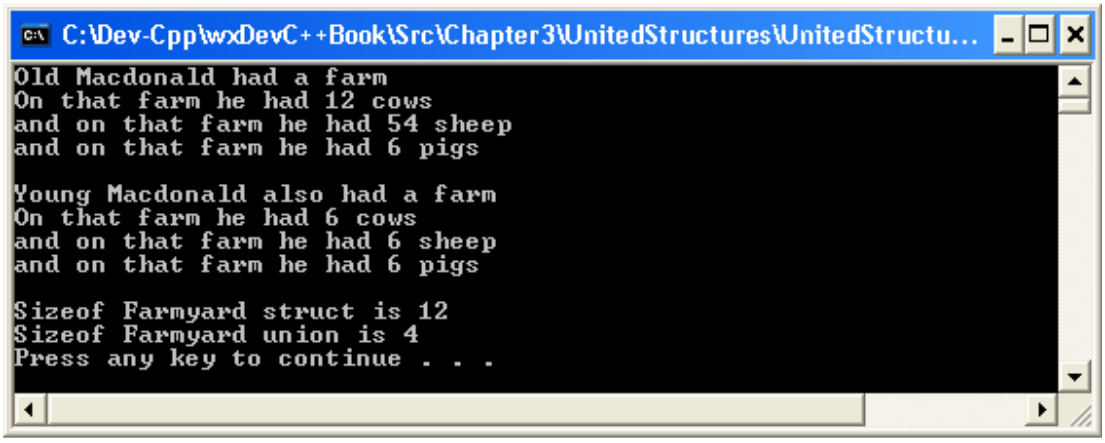


Figure 3.7 – Output from UnitedStructures program

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++'.

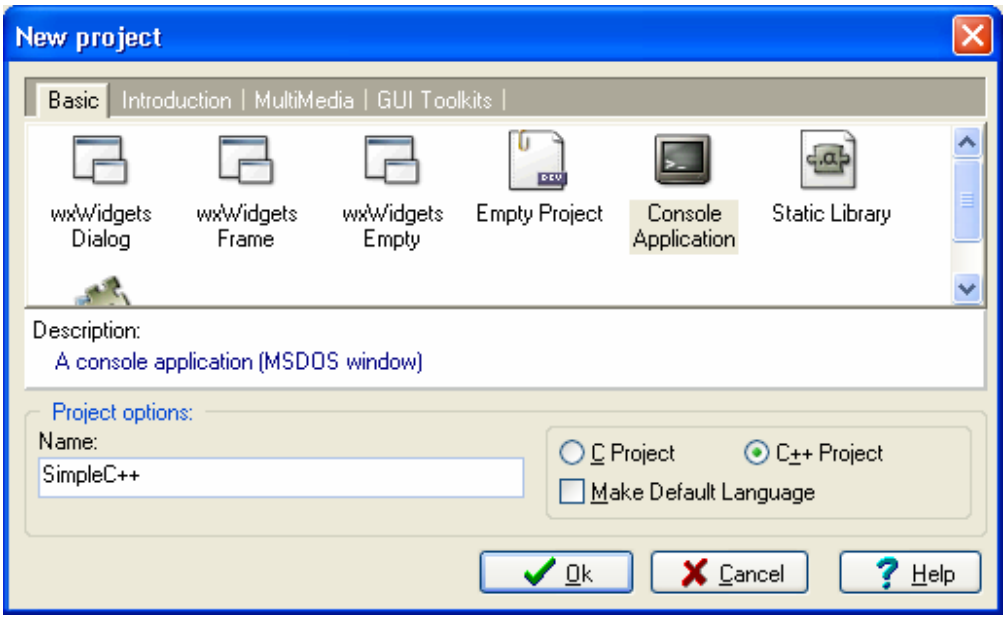


Figure 4.1 – Project settings for new SimpleC++ application.

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;
      int Width;
};
```

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;
   int Width;
};
```

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;
      int Width;
};
```

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.

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.

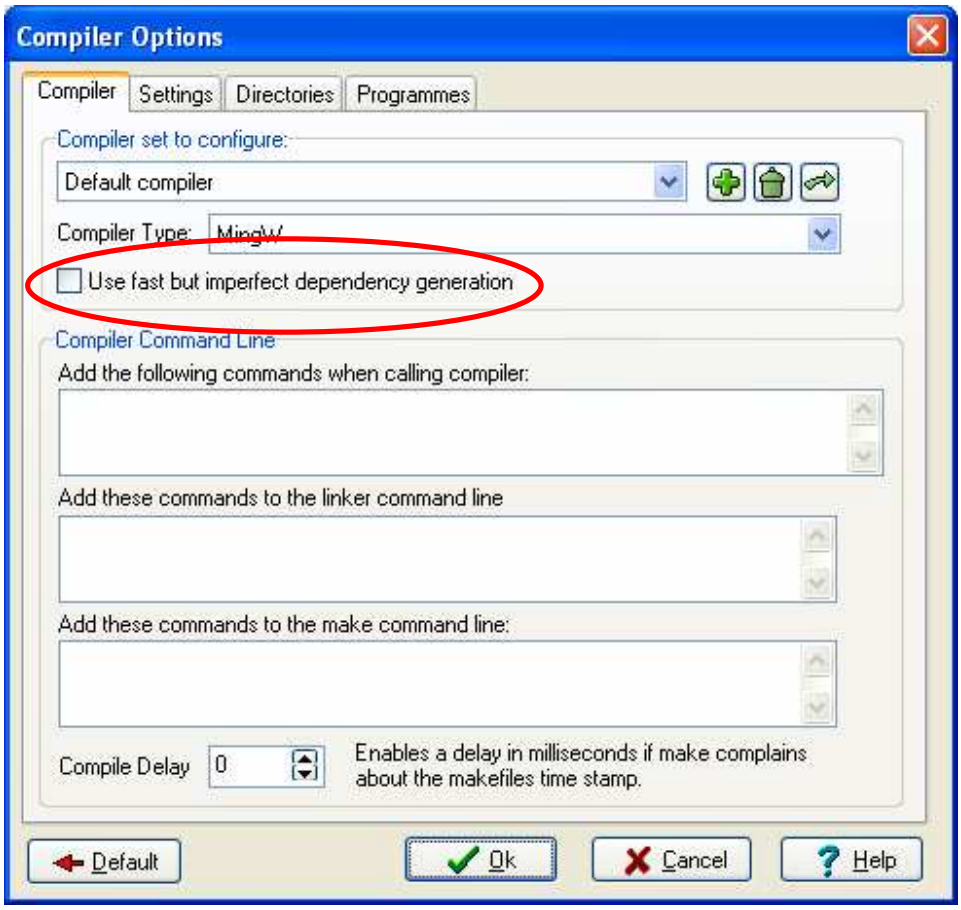


Figure x.x – Use fast but imperfect dependency generation

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.

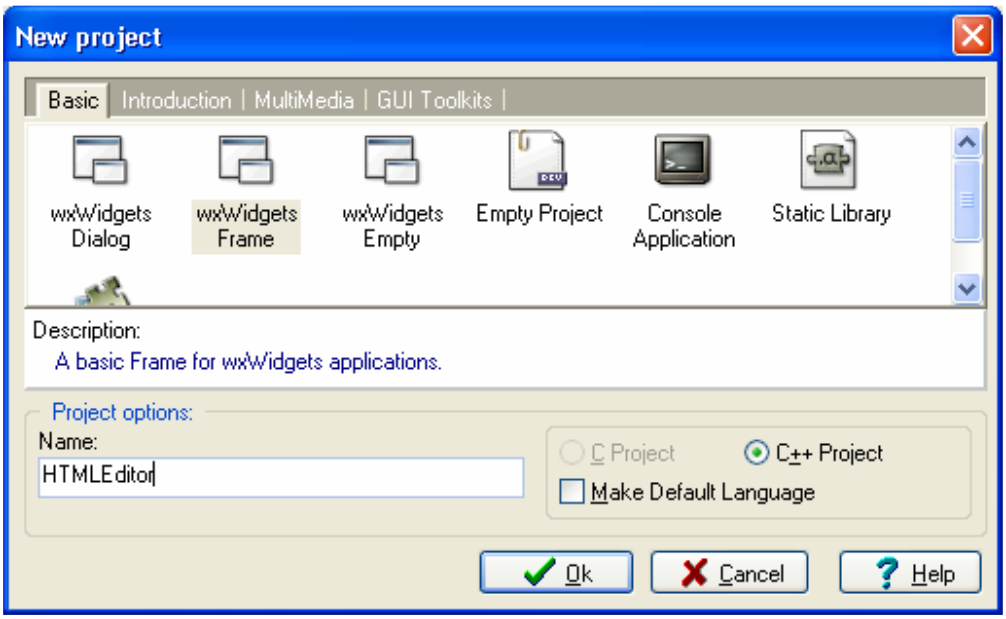


Figure 7.1 – Project settings for HTML editor

Create a new folder called 'HTMLEditor 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.

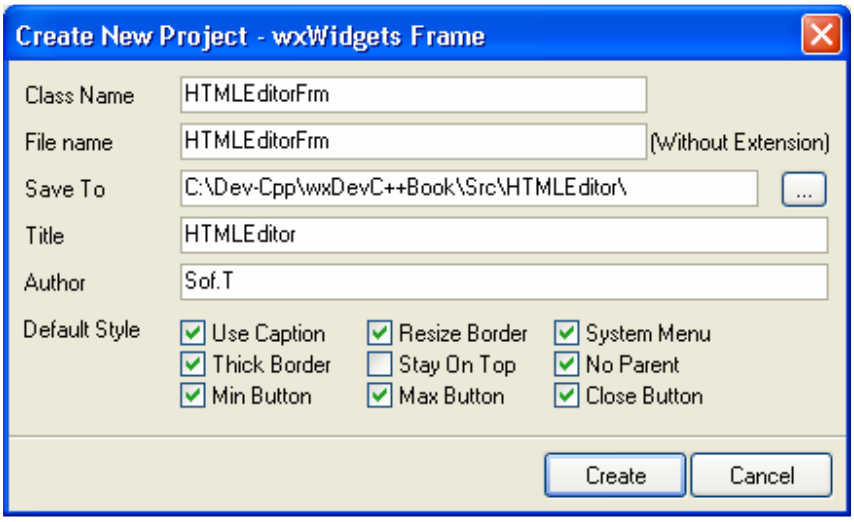


Figure 7.2 – Create New Project dialog

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

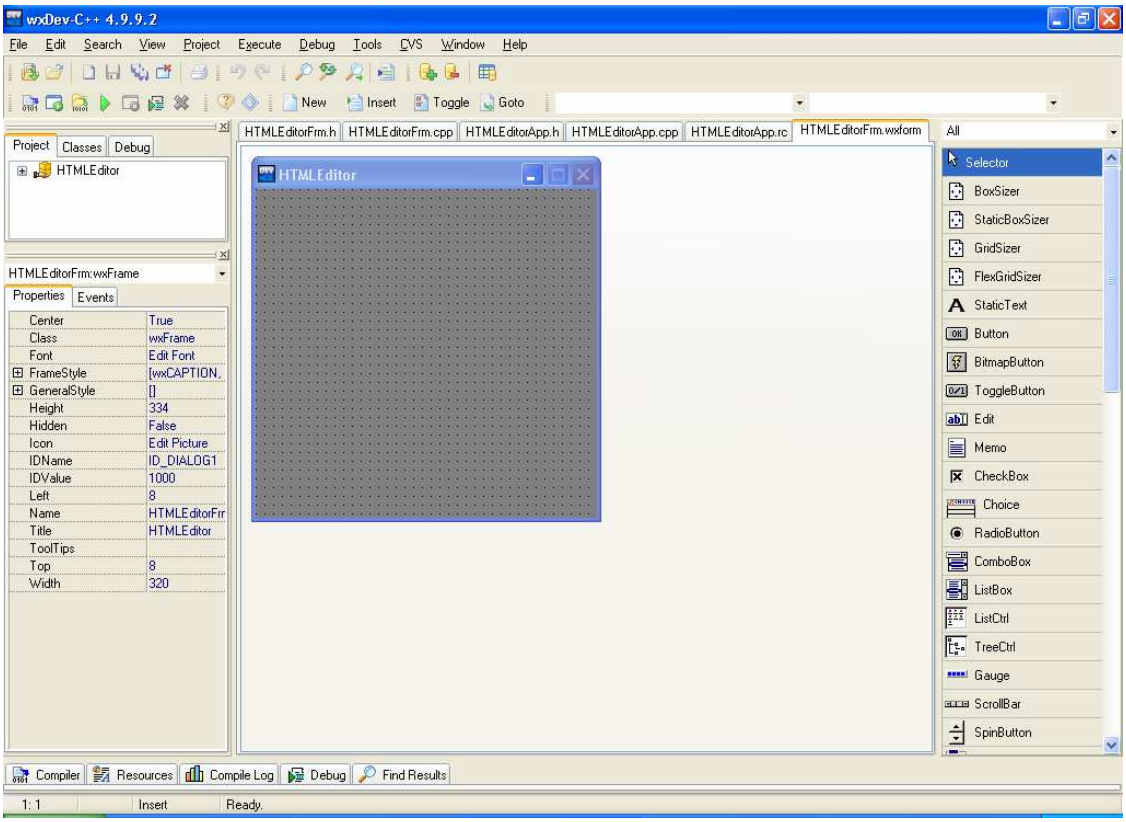


Figure 7.3 – The IDE showing the form designer

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.

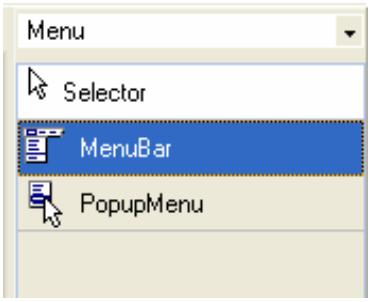


Figure 7.4 – Selecting a menu component

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.

It will automatically find its way under the html window.

Figure 7.5 – 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.

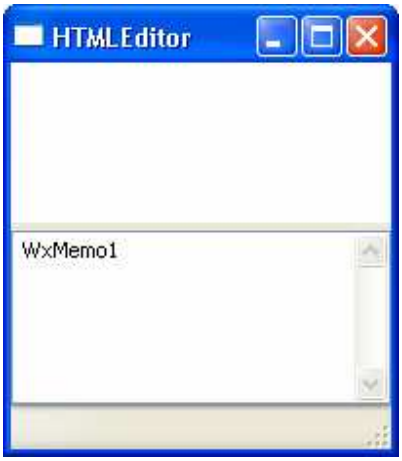


Figure 7.6 – The compiled application.

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 property editor. 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 editor should look like this.

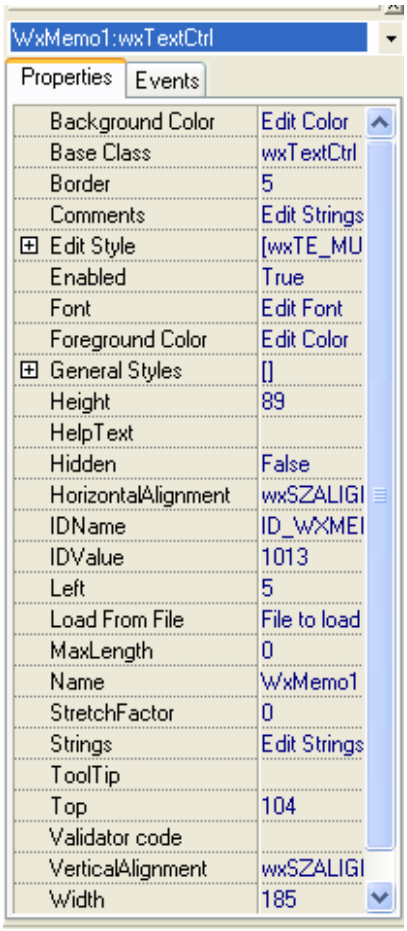


Figure 7.7 – The property editor for the text control

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.

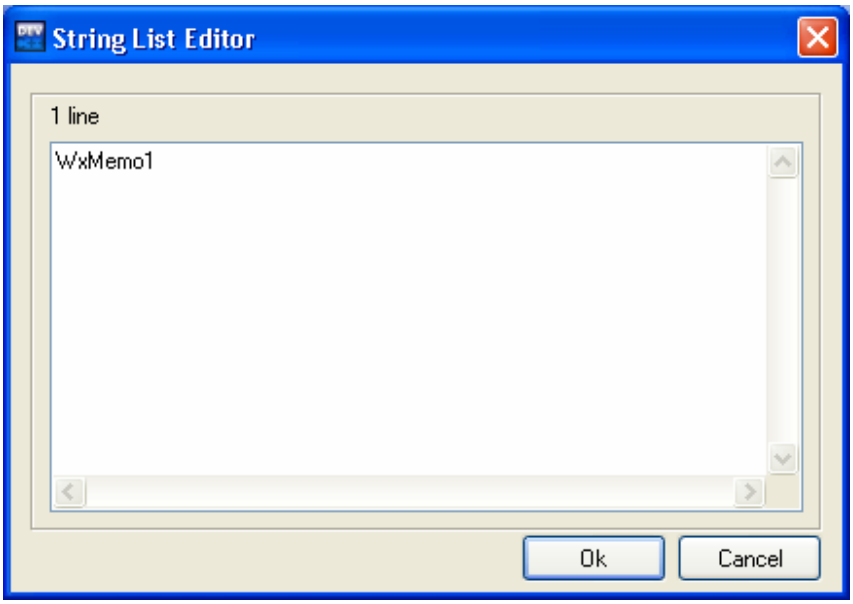


Figure 7.8 – The string editor

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.

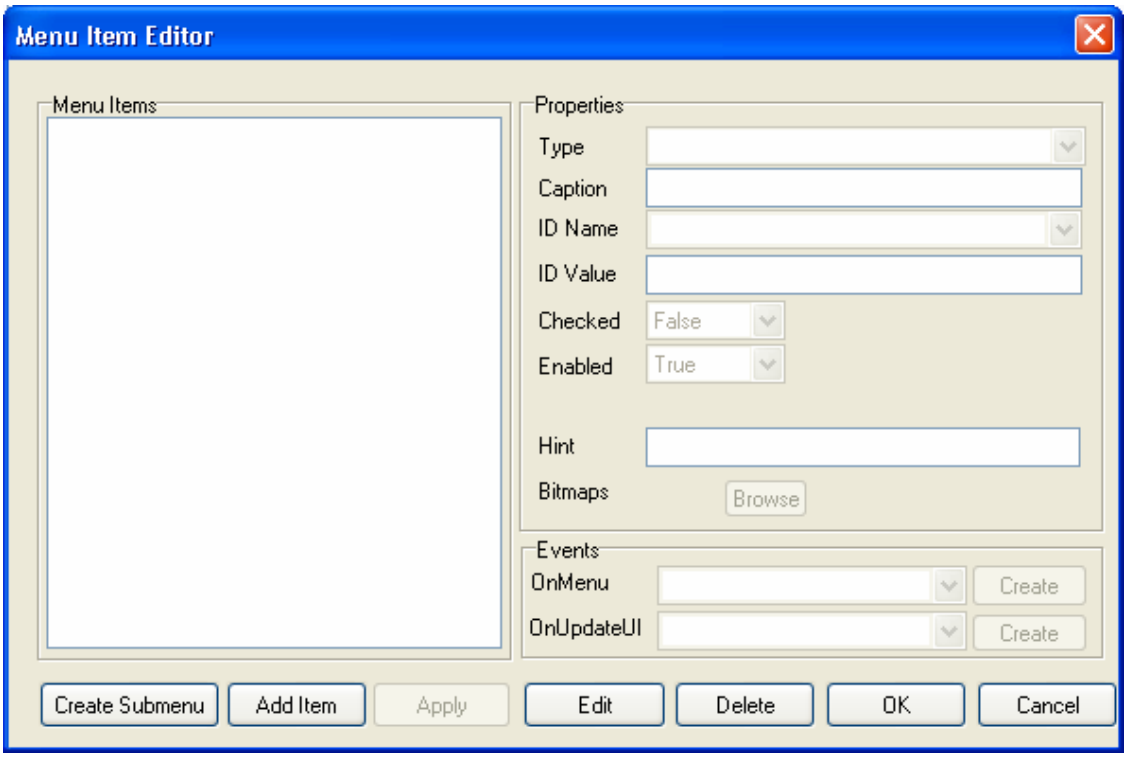


Figure 7.9 – The menu bar editor

Click the [Add Item] button.

Suddenly the dialog fills with all sorts of information that is generated for you. The box labeled ‘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 12 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.

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.

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.

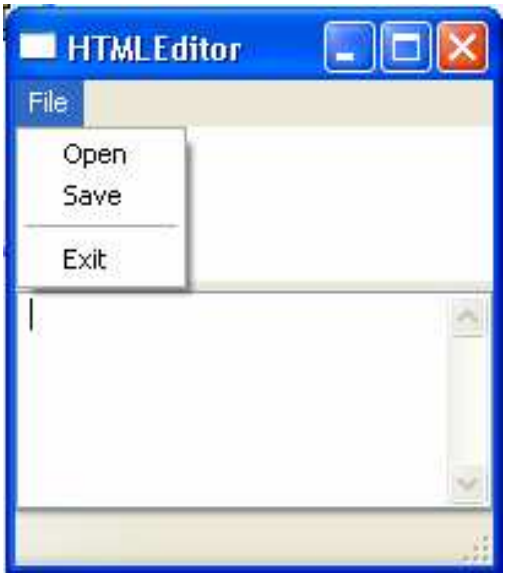


Figure 7.10 – Our new menu bar

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.

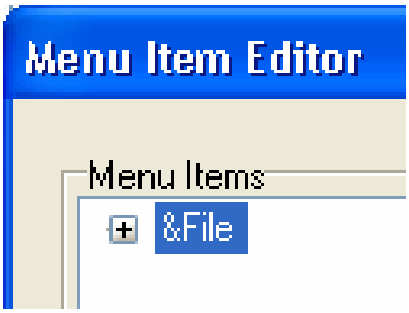


Figure 7.11 – Click the '+' sign to expand the menu tree

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.

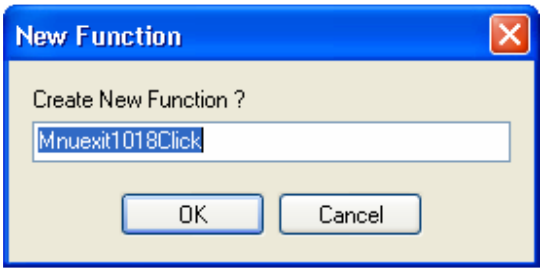


Figure 7.12 – The create new function dialog

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.

```
94      /*
```

```
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.

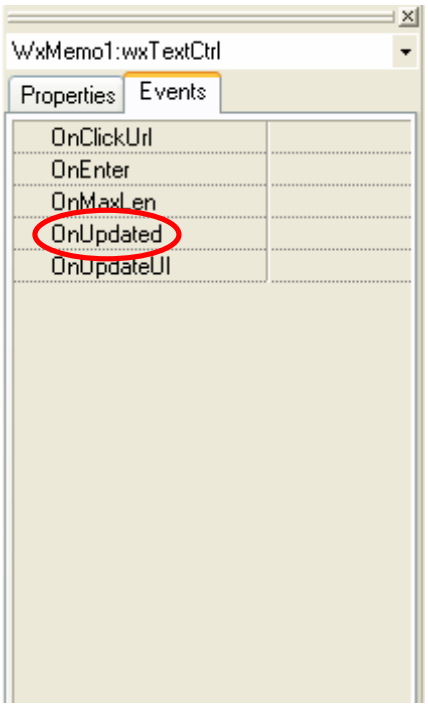


Figure 7.13 – The event editor

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>.

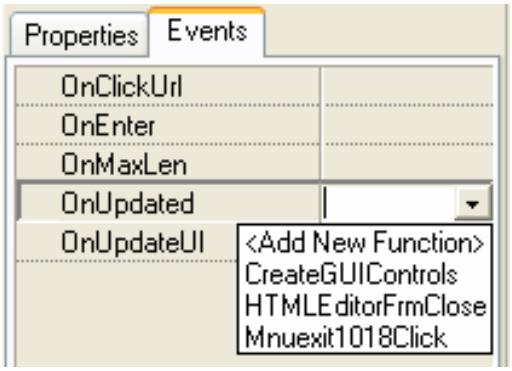


Figure 7.14 – List of available functions in the event editor

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.

```
105    /*
```

```
106     * WxMemo1Updated
107     */
108    void HTMLEditorFrm::WxMemo1Updated(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:

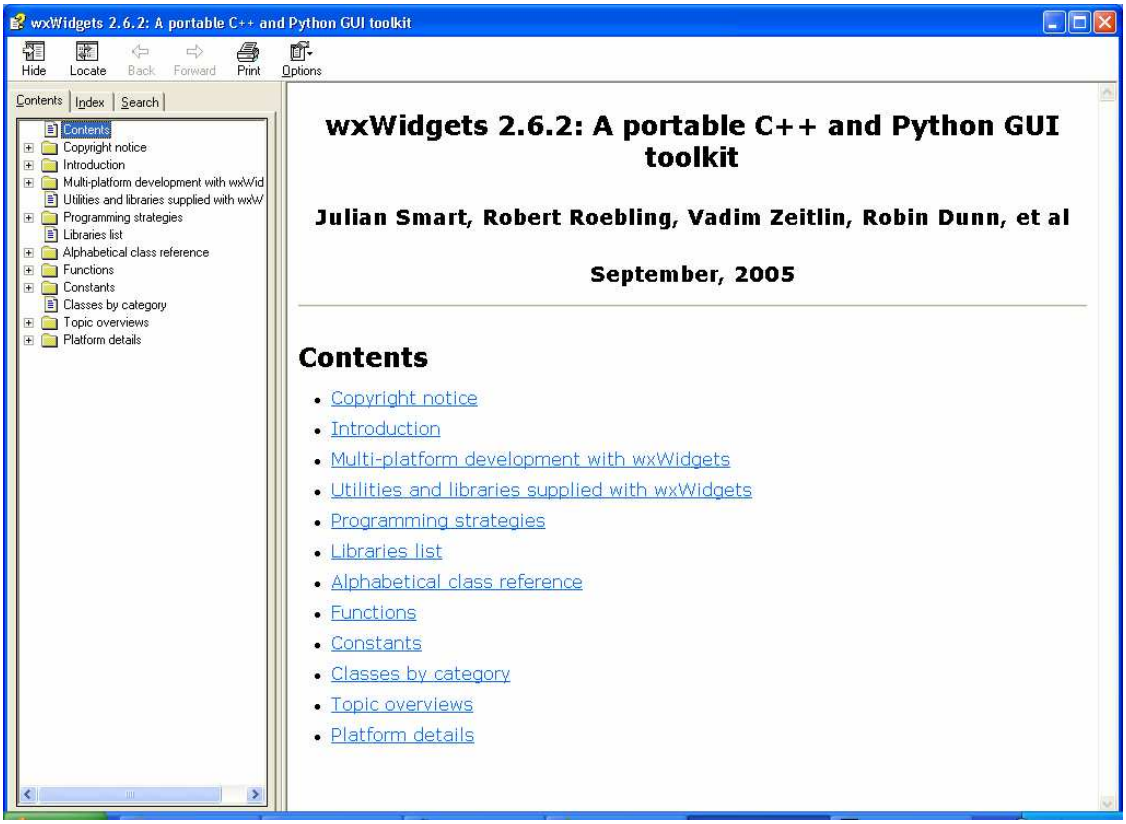


Figure 7.15 – The wxWidgets file help, this should be your greatest friend

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.

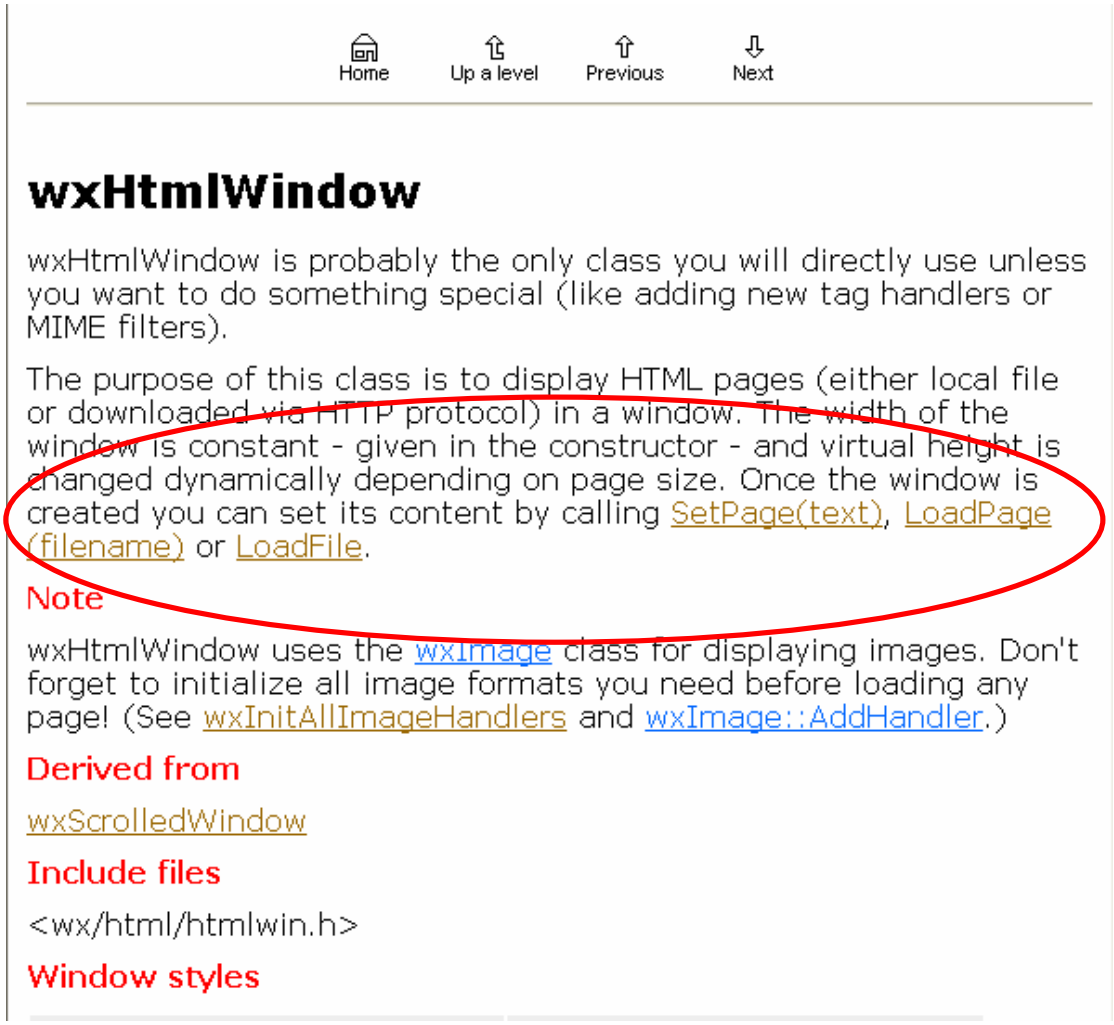


Figure 7.16 – Looking through the description of the wxHtmlWindow

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 7.17 – 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 <b><u>GREAT</u></B> 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            WxMemo1->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            WxMemo1->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 Forms and Dialogs

To be written

Chapter 9 – The Component Palette

Sizers

The components that raises the most questions are the sizers. They have no counterpart in traditional windows programming

Controls

Text Controls
Buttons
Choices
The Rest

Window

Toolbar

Menu

Dialogs

System

Advanced Users

WARNING:	You are advised to back up any files before altering them, then 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 it you could decide what goes where, or have your own sections with your most used components? Well the good news is that you can.

To do this you need to use notepad. Open it then select File|Open. In the Open File dialog make sure the combo box labeled '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'.

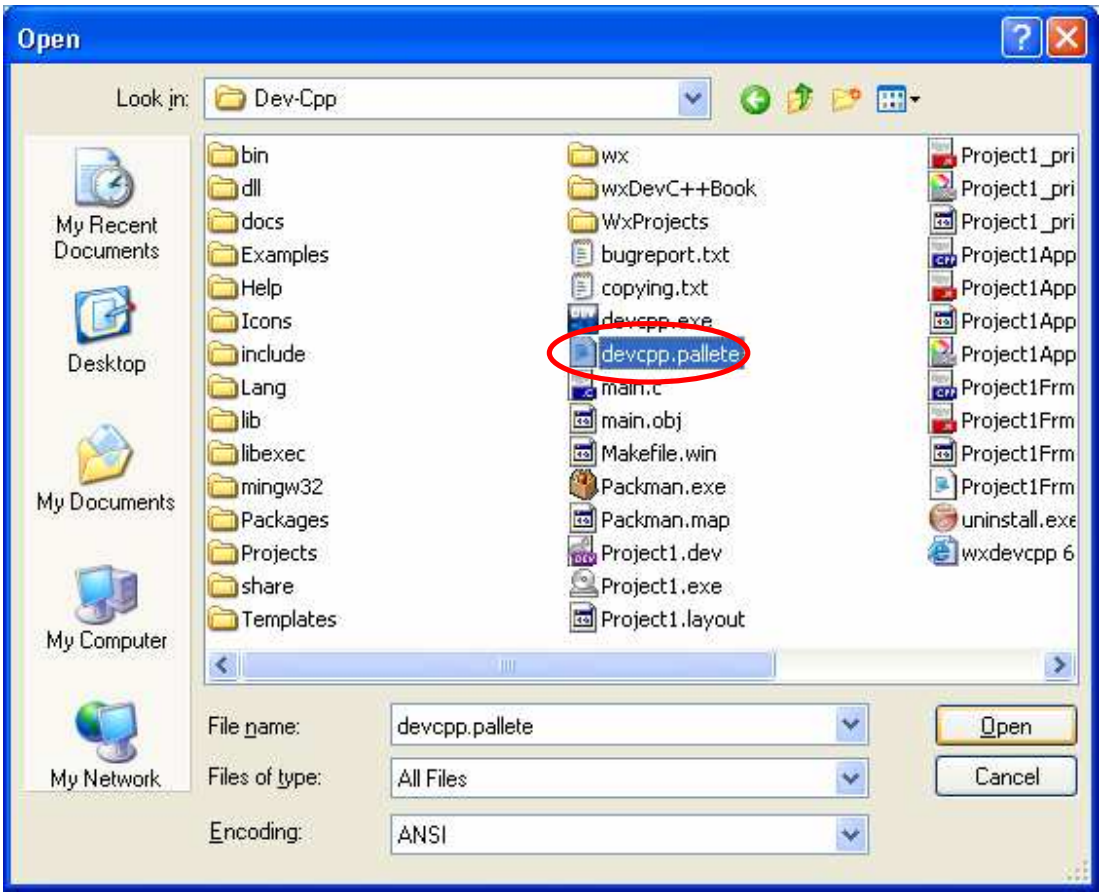


Figure ?.? – Opening devcpp.palette

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;TWxTree
Ctrl;TWxGauge;TWxScrollBar;TWxSpinButton;TWxstaticbox;TWxRadioBox;TWxDatePickerC
trl;TWxSlider;TWxStaticLine;TWxStaticBitmap;TWxStatusBar;TWxChecklistbox;TWxSpin
Ctrl;
Window=TWxPanel;TWxNoteBook;TWxNoteBookPage;TWxGrid;TWxScrolledWindow;TWxHtmlWin
dow;TWxSplitterWindow;
Toolbar=TWxToolBar;TWxToolButton;TWxSeparator;TWxEdit;TWxCheckBox;TWxRadioButton
;TWxComboBox;TWxSpinCtrl;
Menu=TWxMenuBar;TWxPopupMenu;
Dialogs=TWxOpenFileDialog;TWxSaveFileDialog;TWxProgressDialog;TWxColourDialog;TW
xDirDialog;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 think up 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;TWxToolBut
ton;TWxStatusBar;
```

Save the file and then restart wxDev-C++ you should find your new section in the component palette.

Chapter 10 – Linking It With Events

Chapter 11 – Making It Work With Code

Where to add your code (or where did it go)

Chapter 12 – Guidelines for Professional Looking Programs

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.

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 thanks someone for spending the time to make sure that you can operate your computer using only the keyboard.

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.

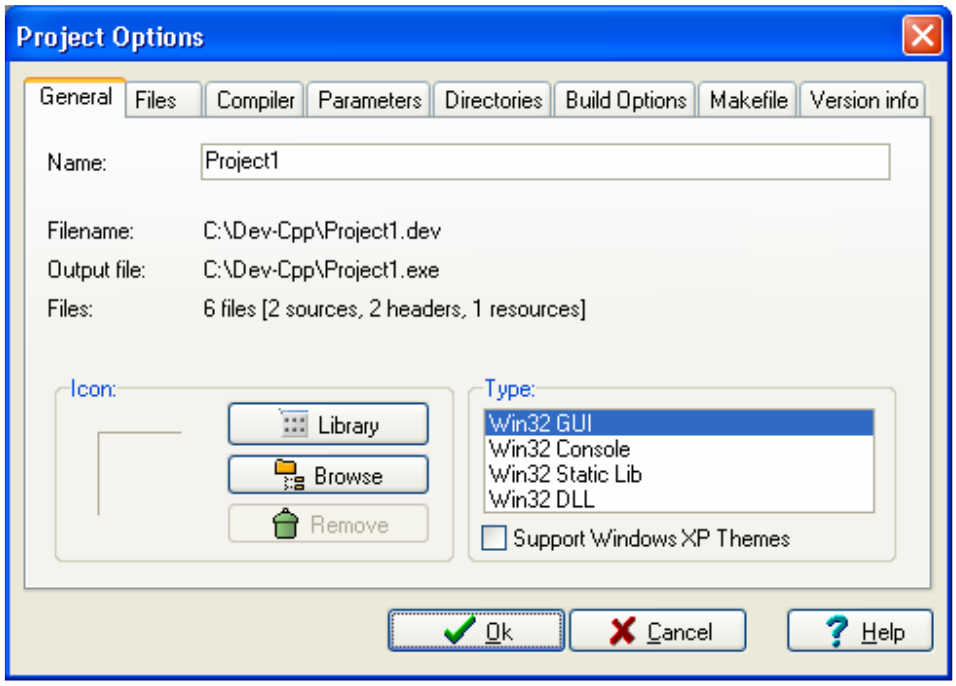


Figure ?.? – The Project Options dialog

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.

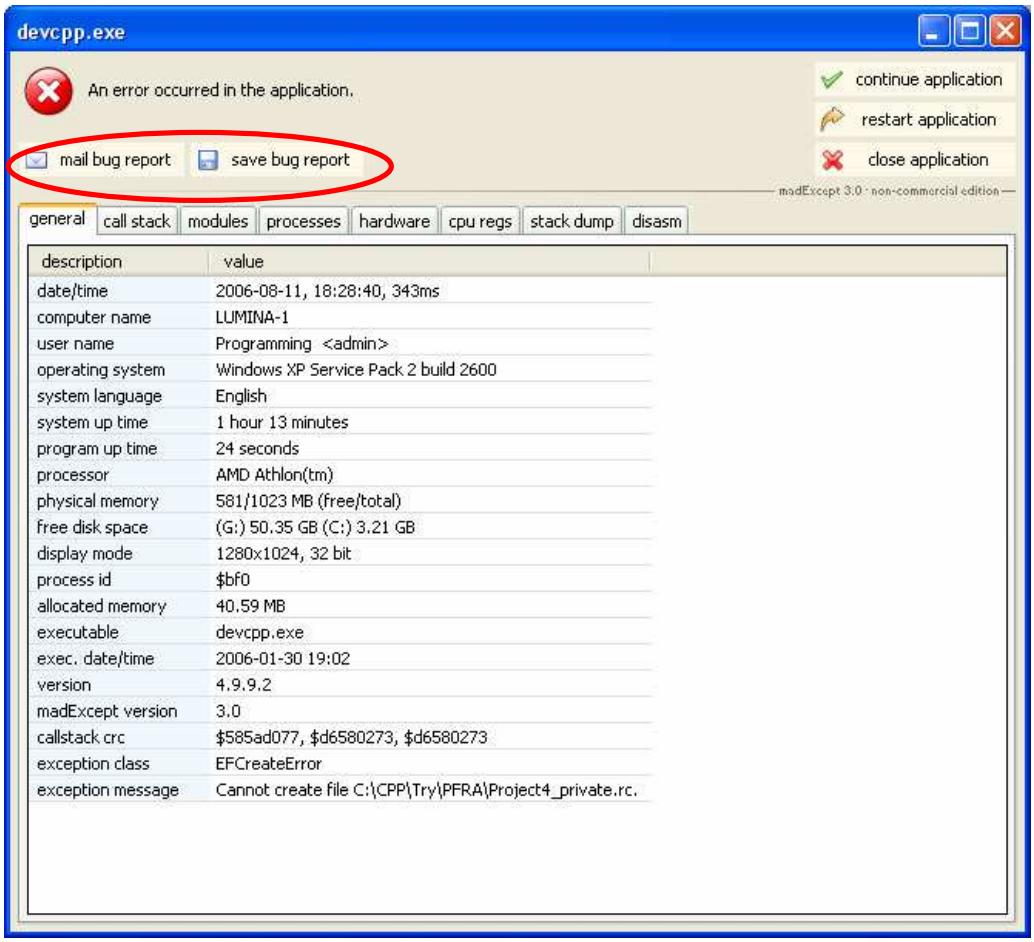


Figure x.x – wxDev-C++ error dialog

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].

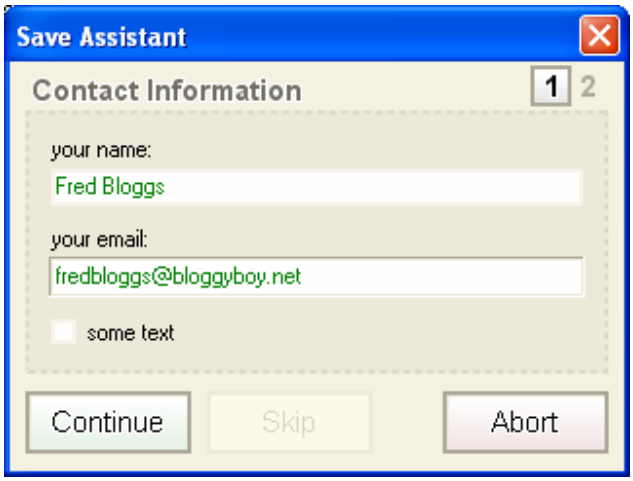


Figure x.x - Step 1, Entering your contact information

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]

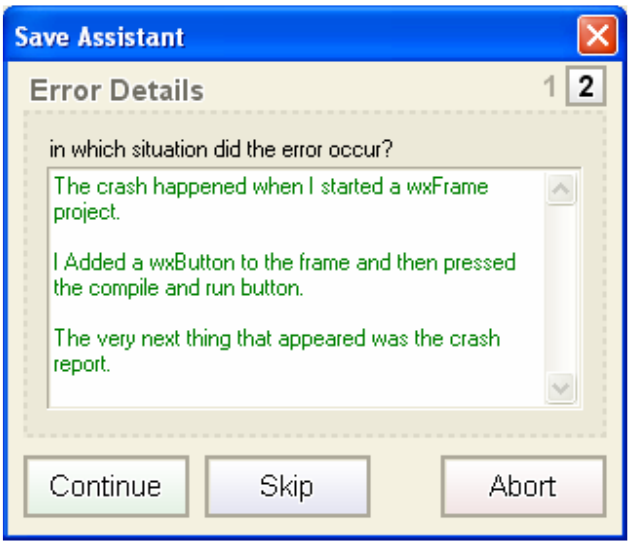


Figure x.x - Step 2, Reporting the steps leading to the crash

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.

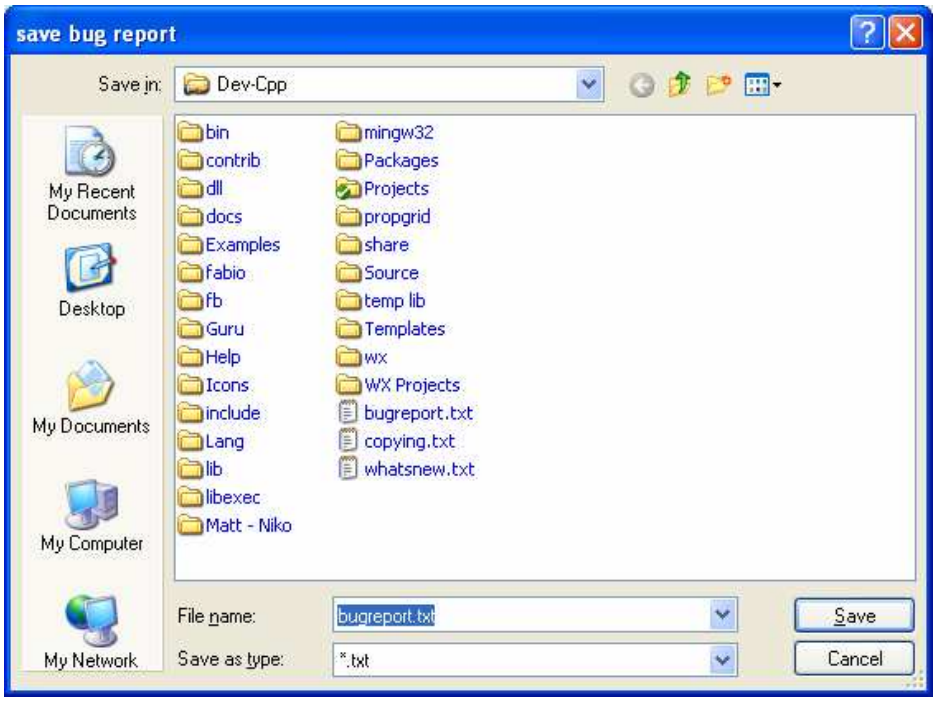


Figure x.x – Step 3, Saving a bug report.

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.

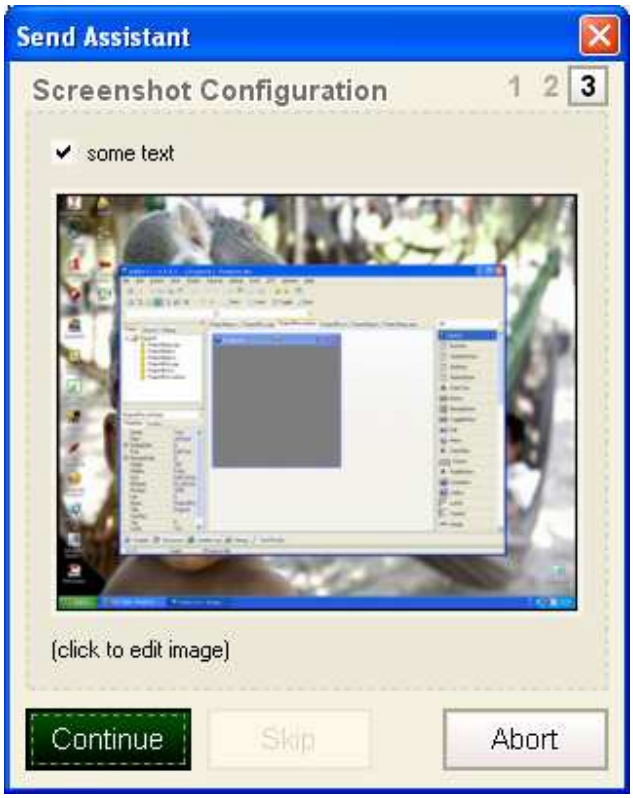


Figure x.x – Screenshot of your desktop and wxDev-C++ running.

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.

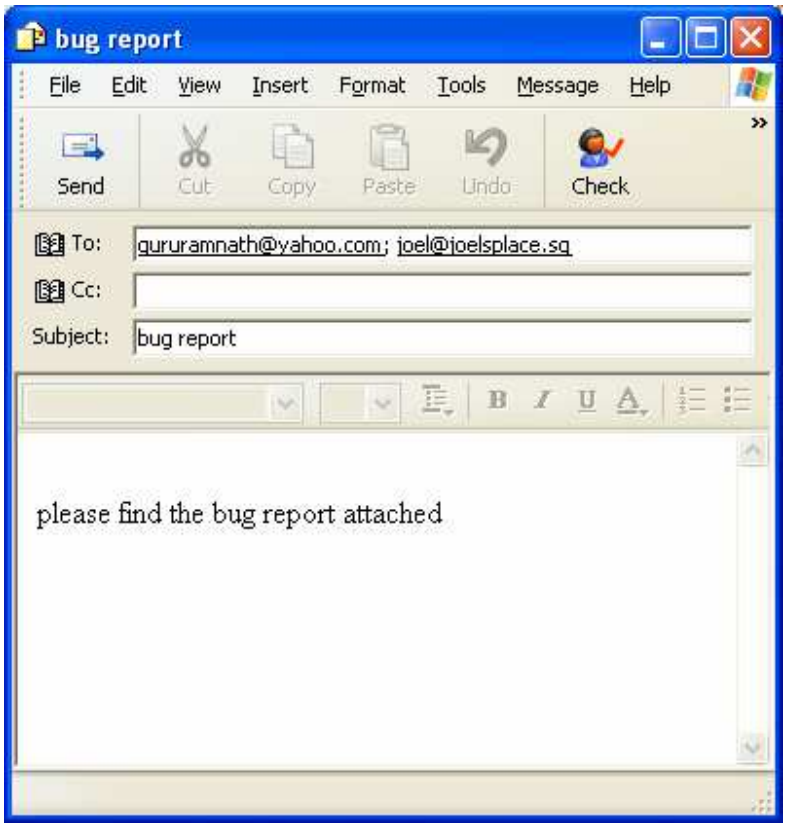


Figure x.x – Email carrying the completed bug report.

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 parameters. 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.

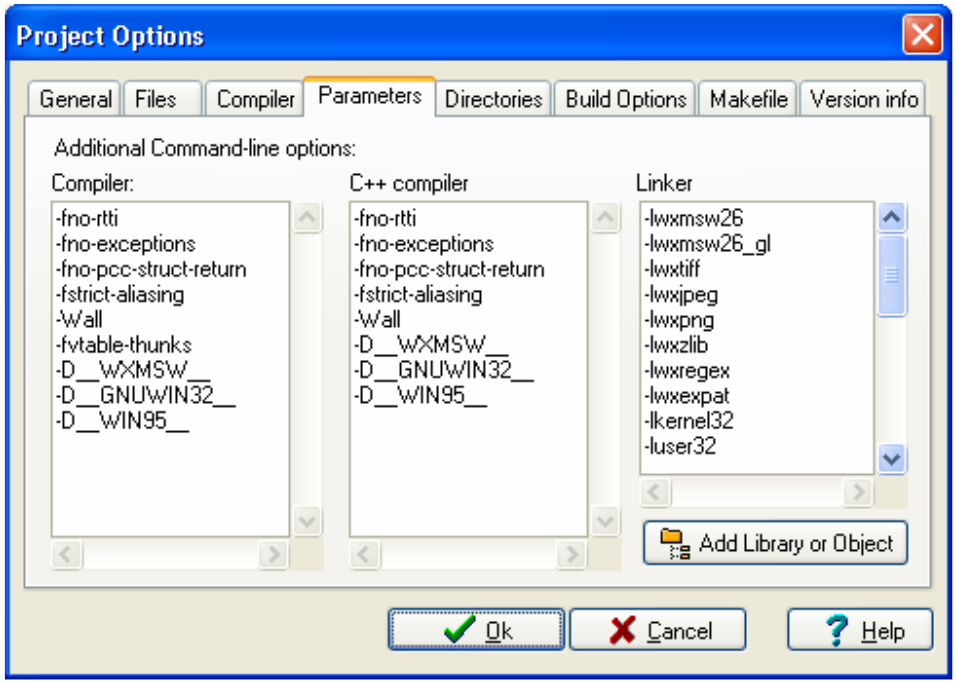


Figure x.x – The Project Options dialog showing wxWidgets 2.6.x library files

Q. I 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'.

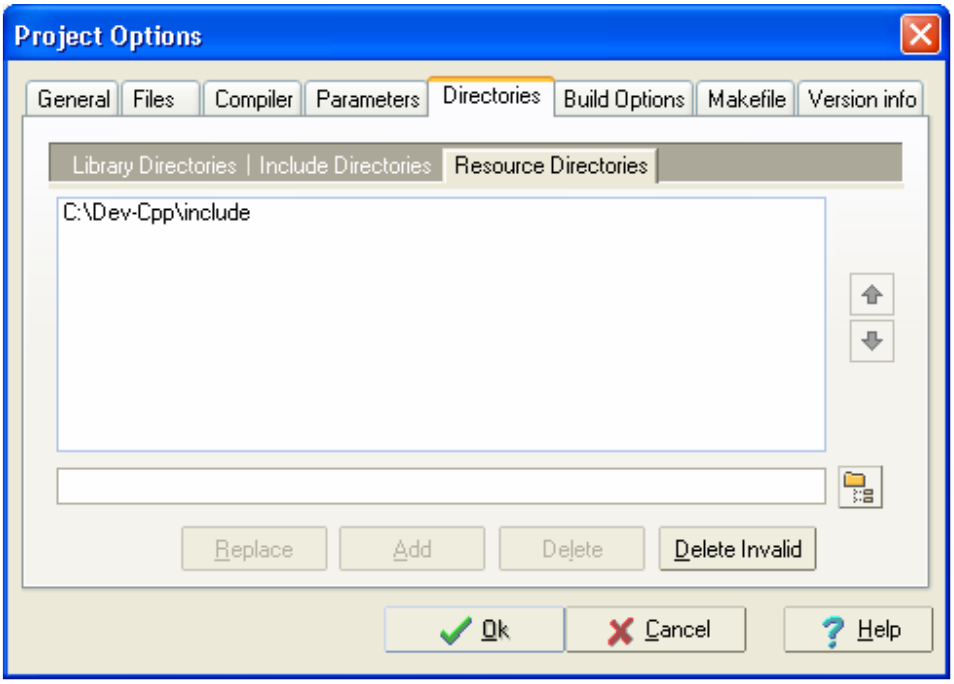


Figure x.x – Project Options dialog showing the resource include path.

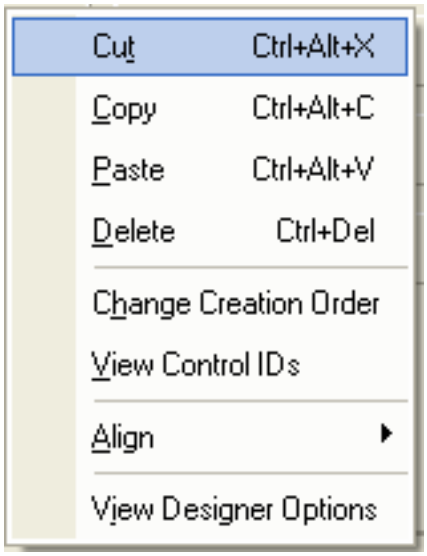
Q. I tried to compile my project but I get the error

In file included from MyProject_private.rc
wx/msw/wx.rc: No such file or directory.

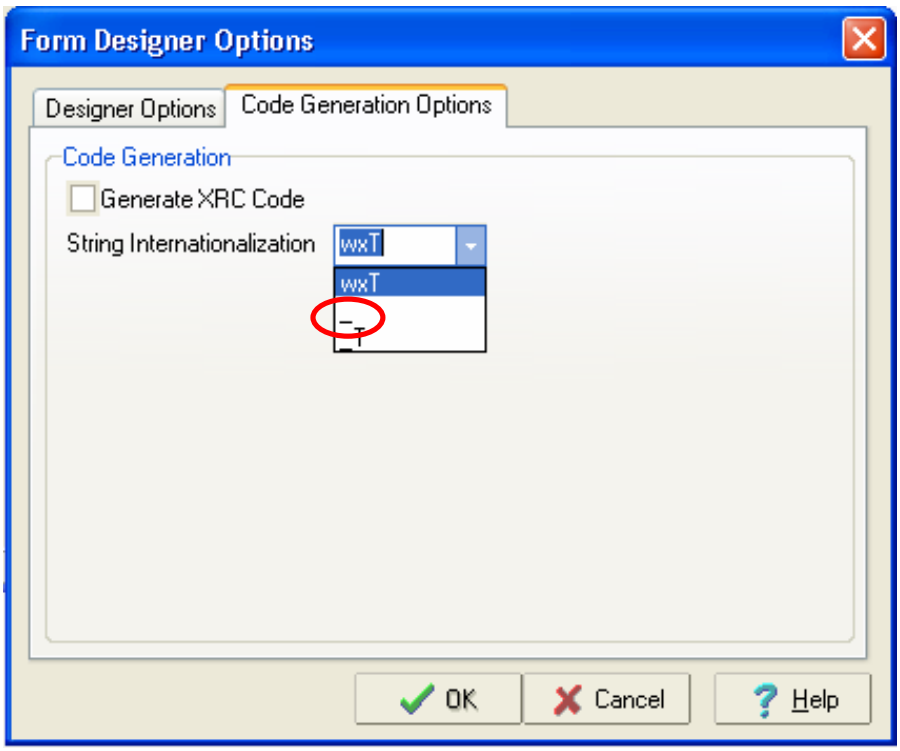
A. This problem is related to the proceeding problem and the answer is the same.

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. 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

C++ Programming

Thinking in C++ Volume 1+2 by Bruce Eckel - 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

C++ Programming - 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

Appendix

Appendix A - Keyboard Shortcuts

Key(s)	Shortcut description
Left Arrow	Left one character
Right Arrow	Right one character
Up Arrow	Up one line
Down Arrow	Down one line
Control + Left	Left one word
Control + Right	Right one word
Home	Start of line
End	End of line
Page Up	Up one page
Page Down	Down one page

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	✓	✓	✓