Writing a simple Cockpit instrument that interfaces to FSX via WideFS / FSUIPC
By David J Ault October 2008 Issue 2

Introduction

First a bit about myself. I have been interested in Flight Simulation since July 2008 when I had the mad idea to build a Learjet Flight Simulator. Being very short of cash I decided to build it as cheap as possible. As I couldn't afford any of the commercial Glass Cockpit software I opted to use a combination of FreeFD and P8R's Excellent Alpha FMC. This worked really well but neither package had the Radio Control Head display so I decided to have a go at writing my own. I had dabbled with a bit of C and Fortran but have never had any programming training so it was quite a steep learning curve.

This tutorial is partly for me to record what I have learnt but mainly to help others who want to have a go at programming their own cockpit instruments. I am not saying that this is the best way to do it, it just shows the way I do it. This tutorial shows how to write a simple Airspeed Indicator for no particular aircraft.

You use this tutorial at your own risk. I take no responsibility for any loss of data or any other problems you may have whilst following this tutorial.

All the tools I use in this tutorial are Freeware. I have quoted version numbers for the software I have used but you may find that there are now later versions.

I have only tried this with XP so not sure if all this works with Vista. Also I have only tried it with FSX but it may work with older versions of Microsoft Flight Simulator.

Setting up the programming environment

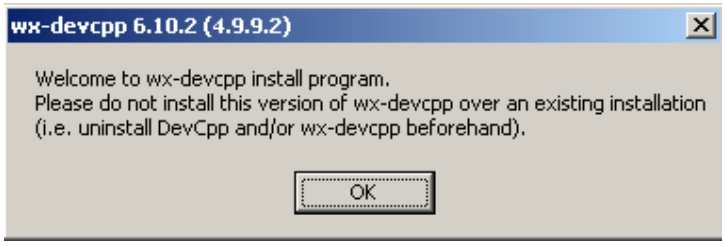
As mentioned above we will be programming using the C language. For this we will use a package that gives you a complete C development environment.

Download the **wxdevcpp_6.10.2_setup.exe** from http://wxdsgn.sourceforge.net/ This is based on MingW which is a windows port of the GCC Compiler and contains an Editor, Compiler &Linker.

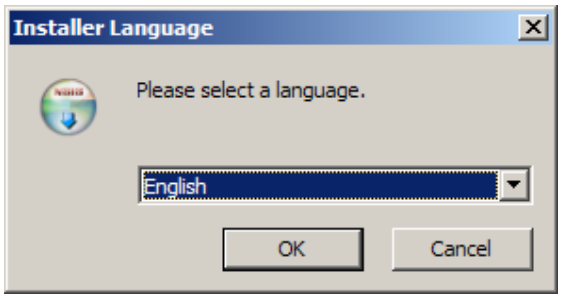
Install as follows:-

Run **wxdevcpp_6.10.2_setup.exe**

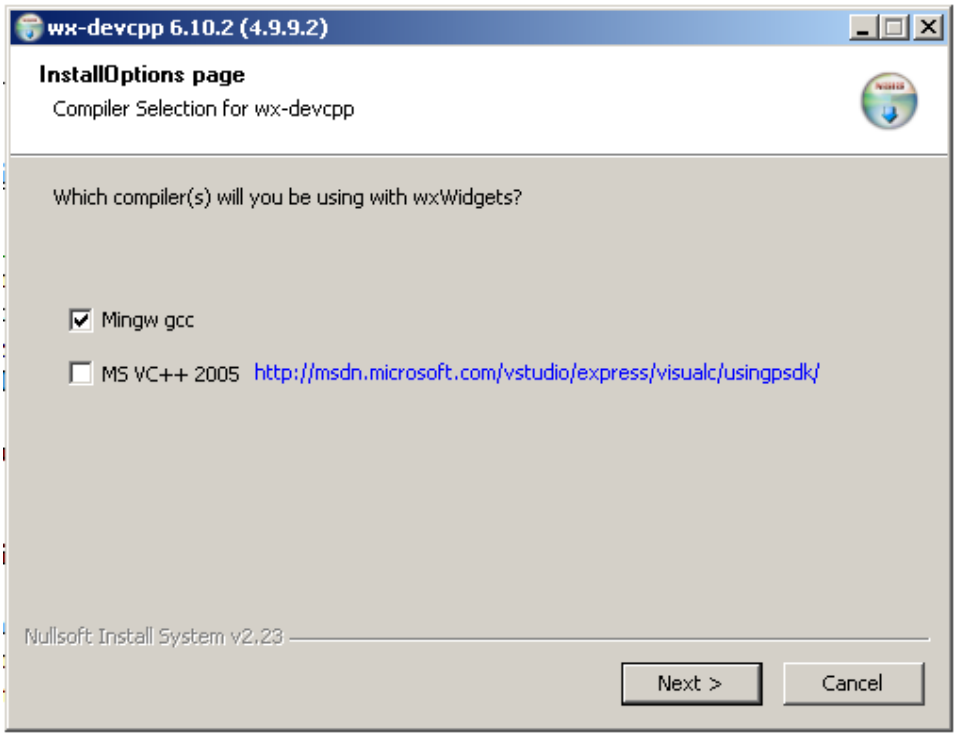
Assuming that you have not already installed a copy of wx-devcpp, press **OK**.



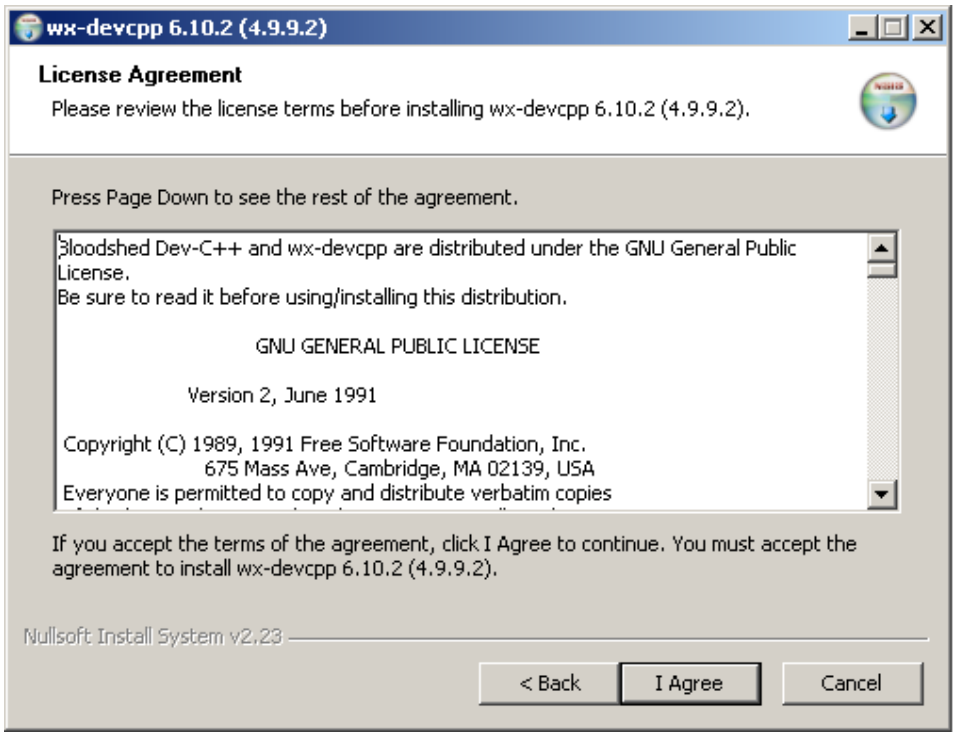
Select your language and press **OK**



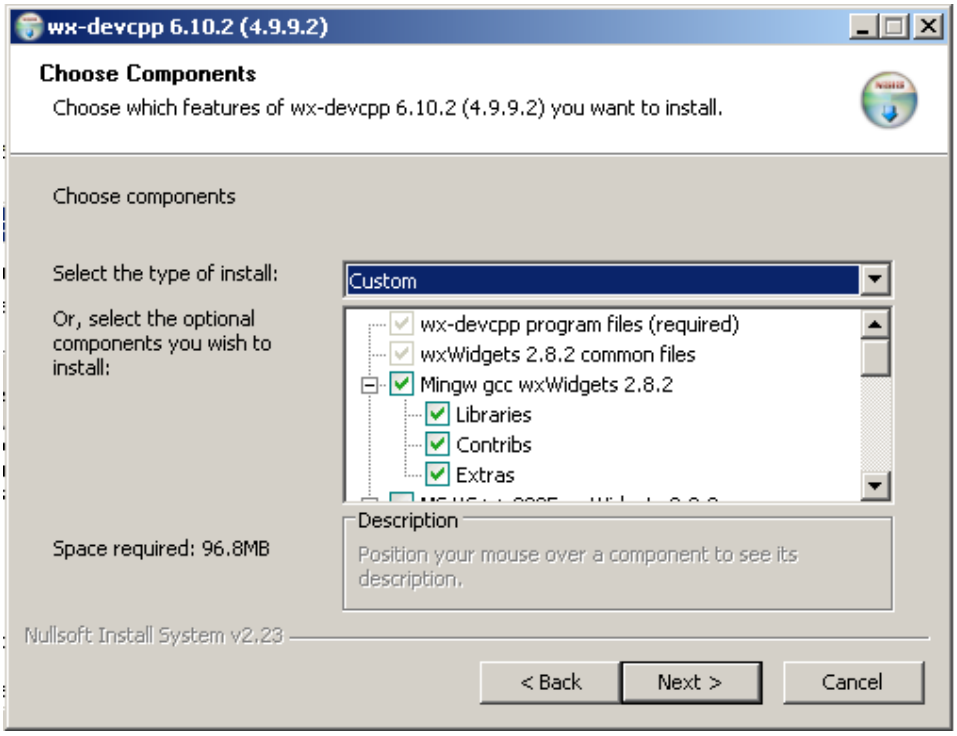
Ensure that only **Mingw gcc** is selected and press **Next**.



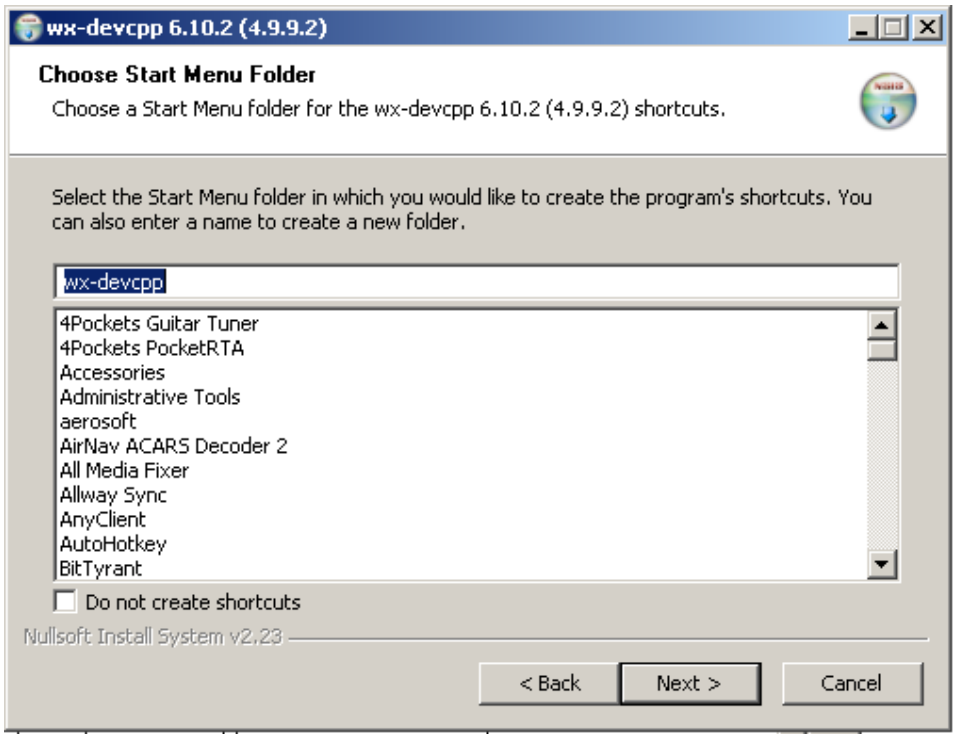
Read the License Agreement and click **I Agree** if happy to continue.



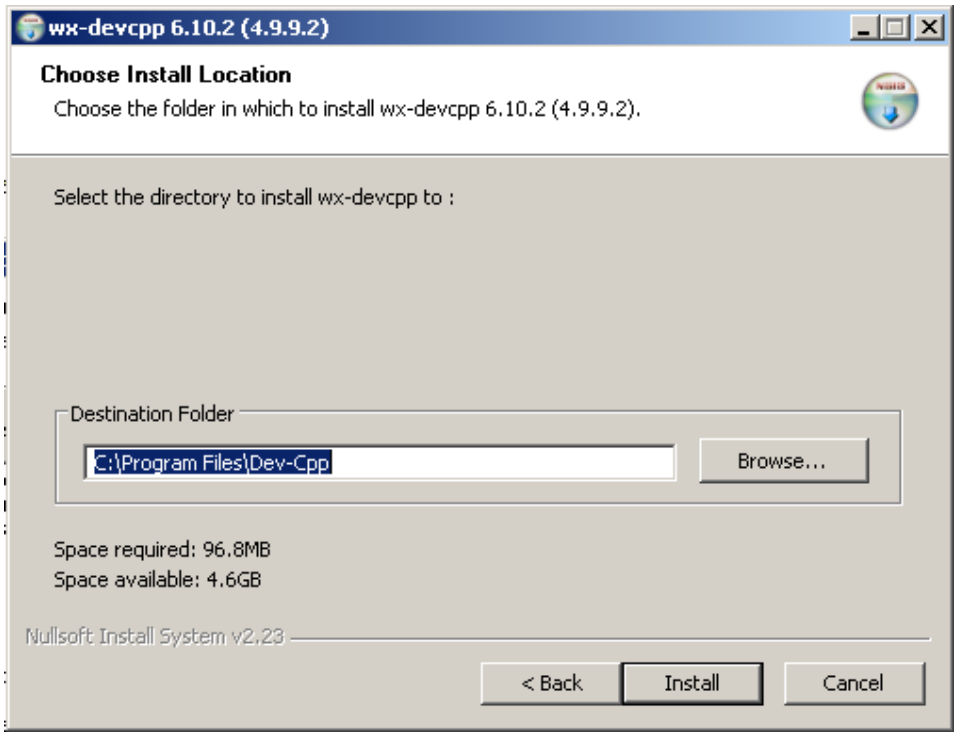
Leave **Type of Install** as **Custom** and click **Next**.



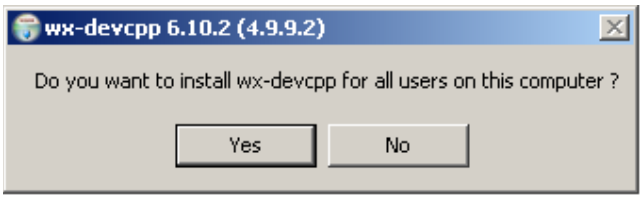
At **Choose Start Menu Folder**, click **Next**.



Select where you want to install it.
For this tutorial I have left it as C:\Program Files\Dev-Cpp

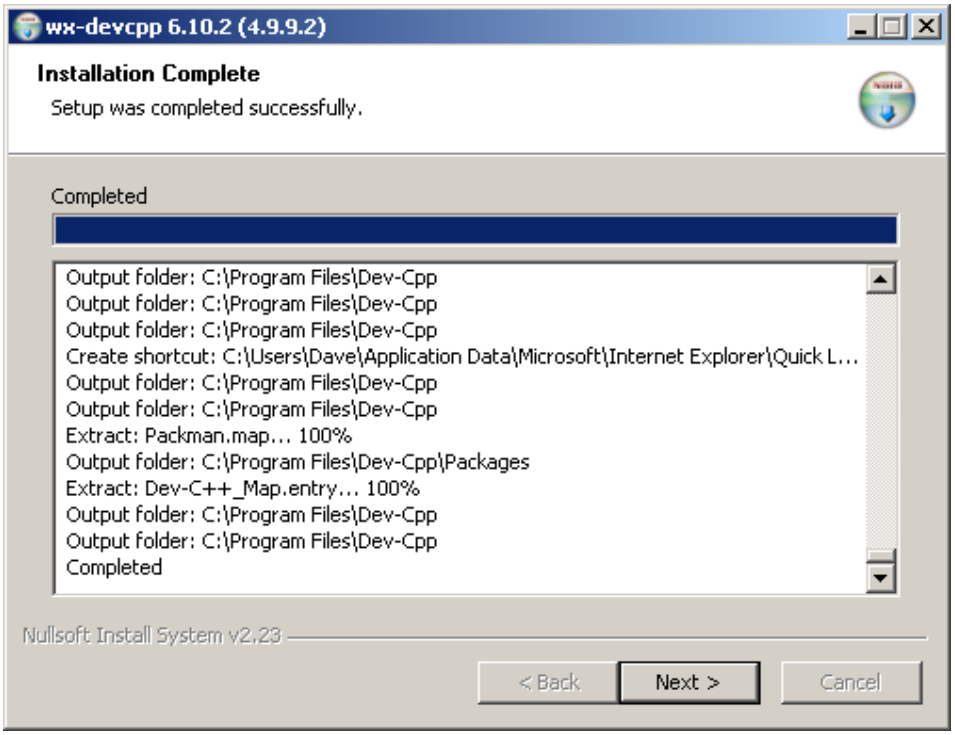


The program will install, then ask

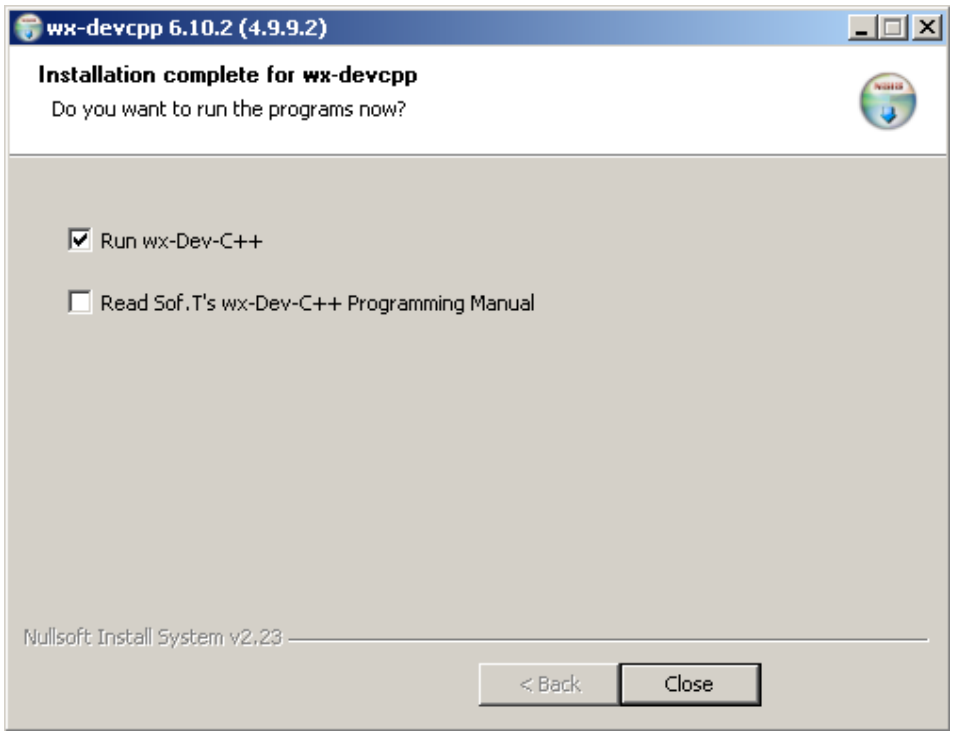


I selected **Yes**

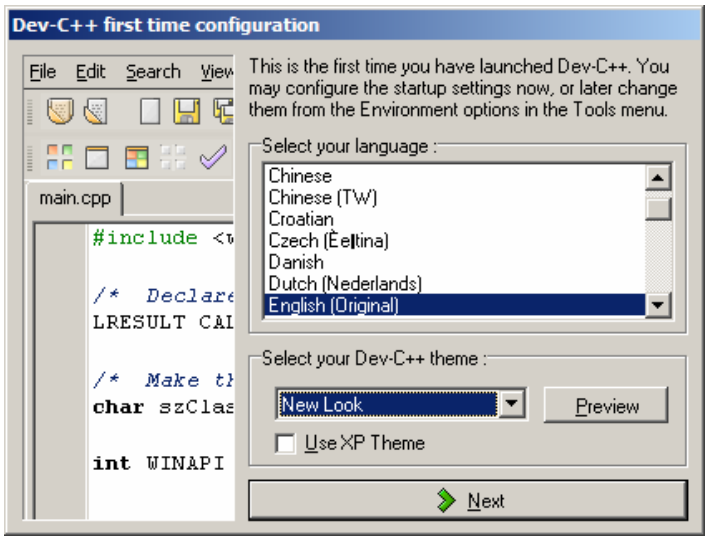
At the next screen click **Next**

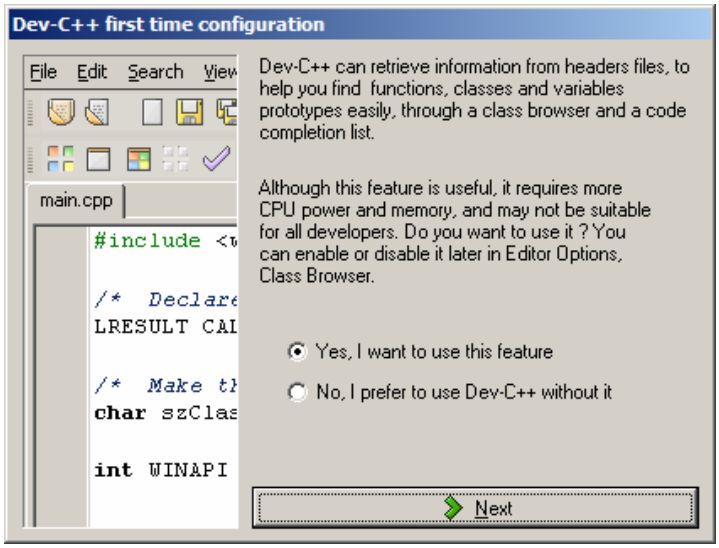


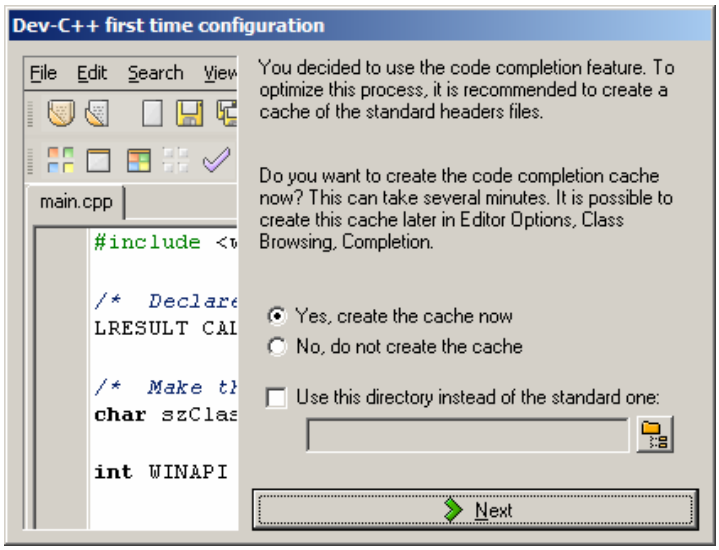
Untick **Read Sof.T's wx-Dev-C++ Programming Tutorial** and click on **Close**.



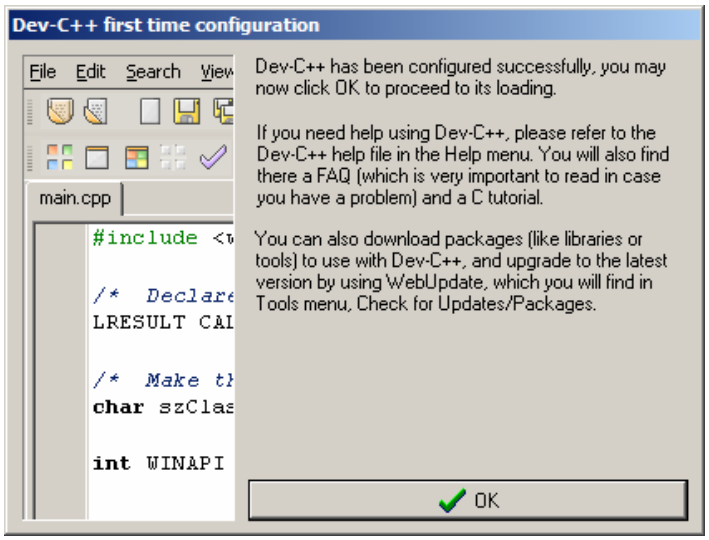
Wx-DevCpp will now run and ask you a few questions. Just click next to the following:-



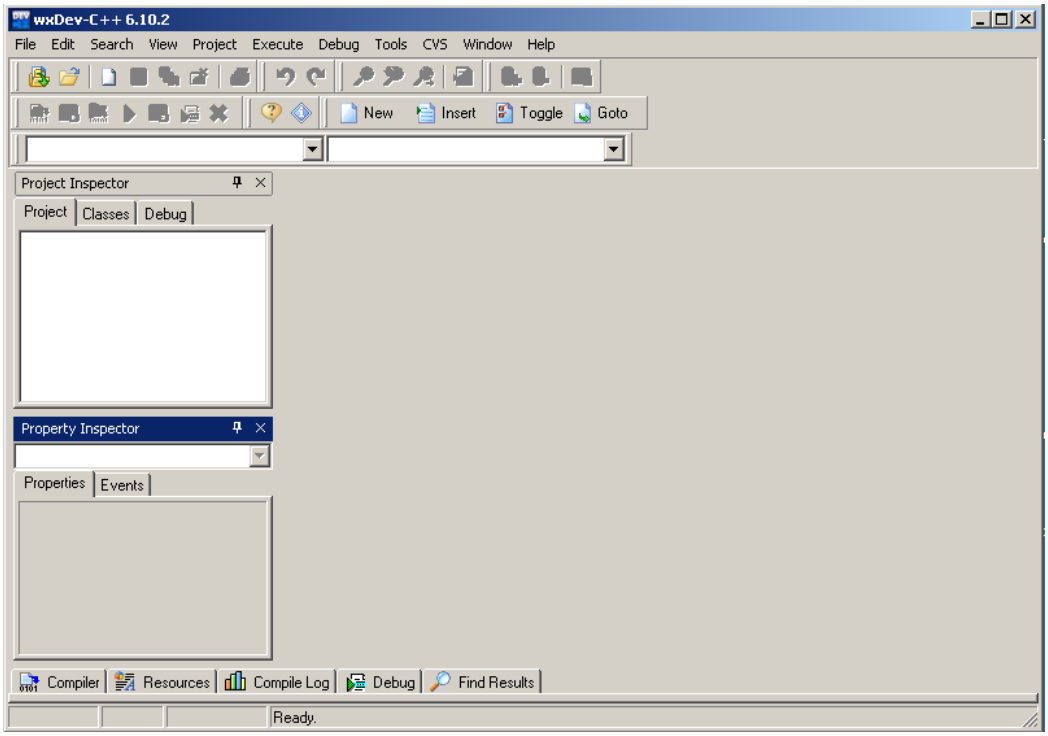

Dev-C++ first time configuration
File Edit Search View
main.cpp
#include <w
/* Declare
LRESULT CAL
/* Make th
char szClas
int WINAPI
Dev-C++ can retrieve information from headers files, to help you find functions, classes and variables prototypes easily, through a class browser and a code completion list.
Although this feature is useful, it requires more CPU power and memory, and may not be suitable for all developers. Do you want to use it ? You can enable or disable it later in Editor Options, Class Browser.
Yes, I want to use this feature
No, I prefer to use Dev-C++ without it
Next


Dev-C++ first time configuration
File Edit Search View
main.cpp
#include <w
/* Declare
LRESULT CAL
/* Make th
char szClas
int WINAPI
You decided to use the code completion feature. To optimize this process, it is recommended to create a cache of the standard headers files.
Do you want to create the code completion cache now? This can take several minutes. It is possible to create this cache later in Editor Options, Class Browsing, Completion.
Yes, create the cache now
No, do not create the cache
Use this directory instead of the standard one:
Next

Click **OK**


Dev-C++ first time configuration
File Edit Search View
main.cpp
#include <w
/* Declare
LRESULT CAL
/* Make th
char szClas
int WINAPI
Dev-C++ has been configured successfully, you may now click OK to proceed to its loading.
If you need help using Dev-C++, please refer to the Dev-C++ help file in the Help menu. You will also find there a FAQ (which is very important to read in case you have a problem) and a C tutorial.
You can also download packages (like libraries or tools) to use with Dev-C++, and upgrade to the latest version by using WebUpdate, which you will find in Tools menu, Check for Updates/Packages.
OK

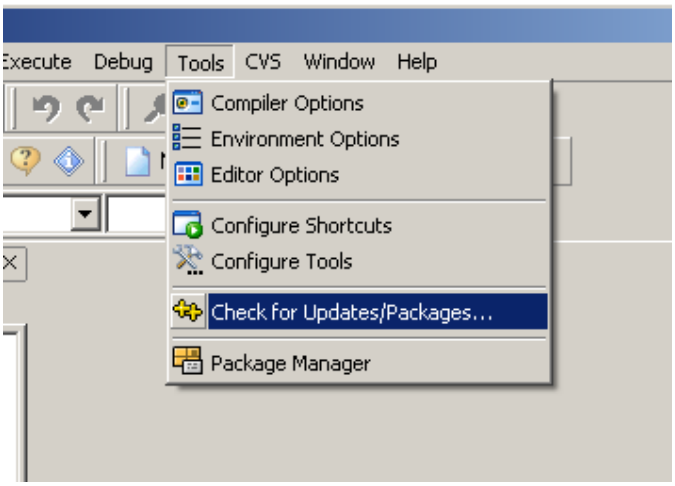
wxDevC++ will now run. Close the ‘Tip of the day’ box and you should see the following:-



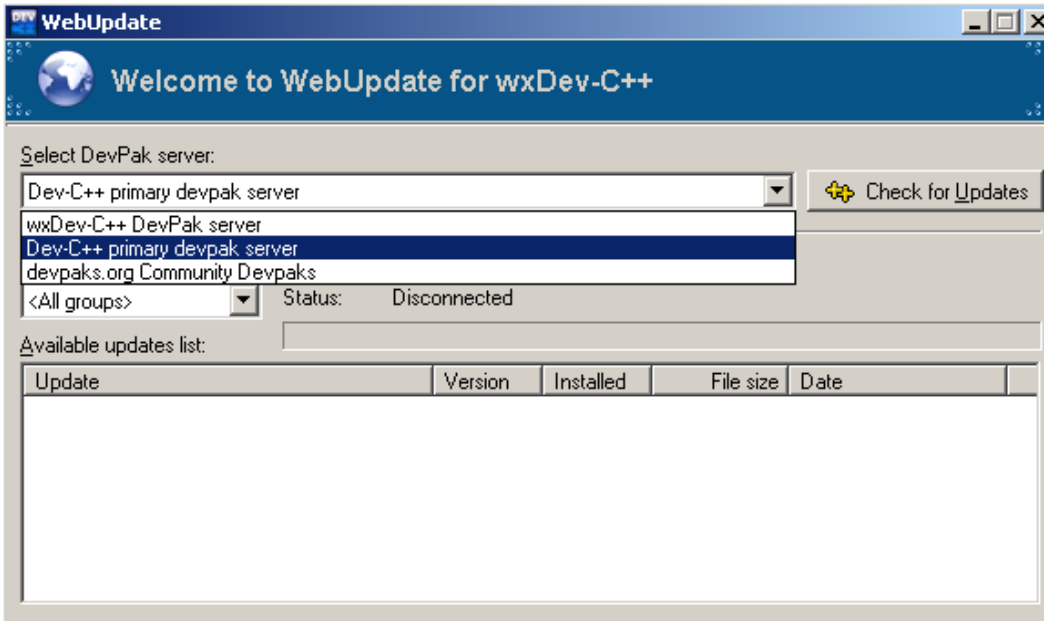
Installing Allegro

Next we will install Allegro which is a games programming library that we will be using for the graphics.

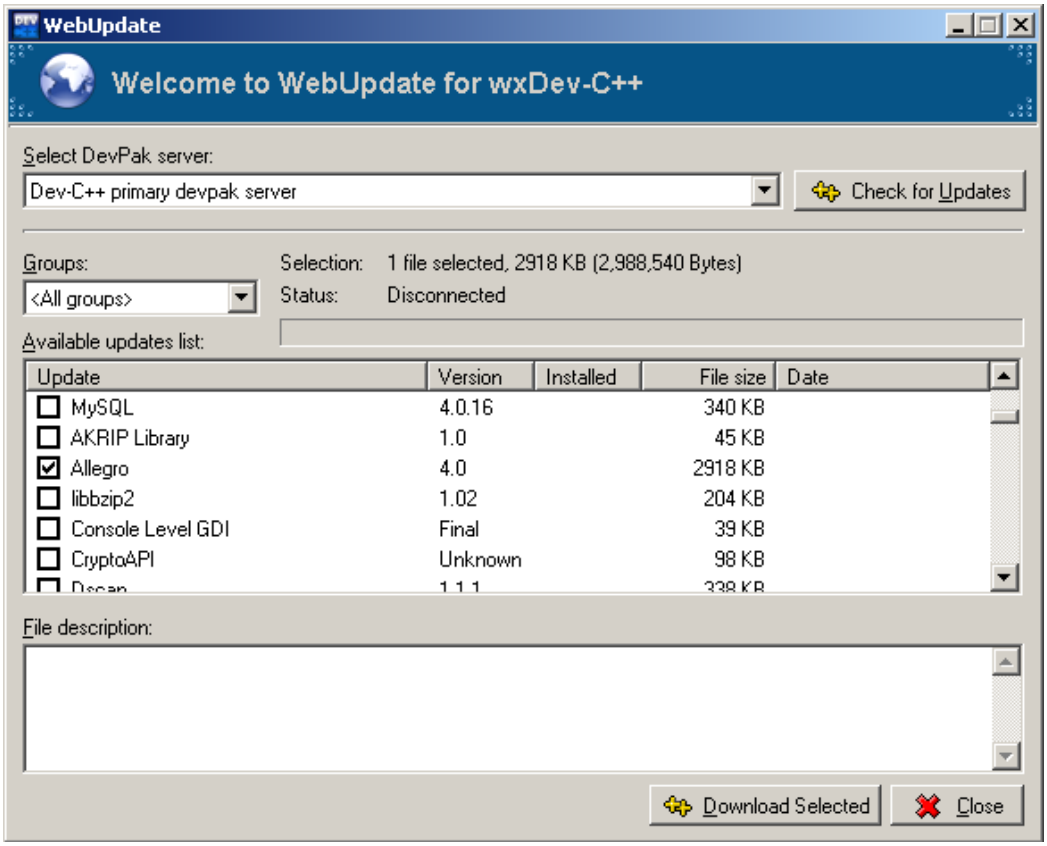
Click **Tools** – **Check for Updates/Packages**



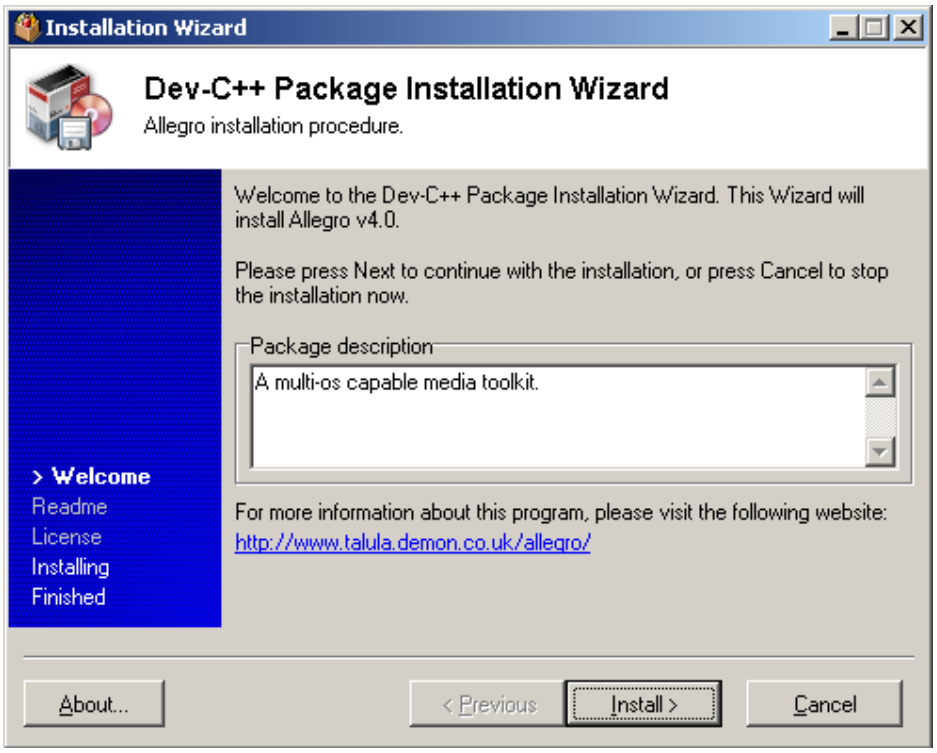
Select **Dev-C++ primary devpak server** then click **Check for Updates**.



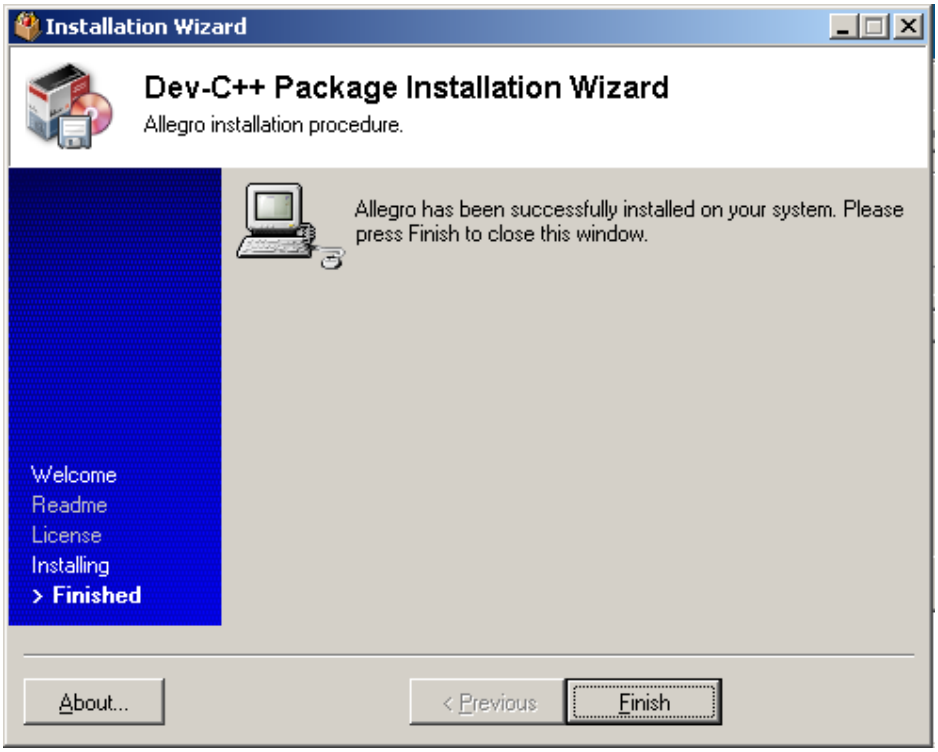
In the **Update List**, Tick **Allegro** then click **Download Selected**.



When prompted, Click **Install >**



When installation has completed click Finish. Then close the WebUpdate Window.



Your programming environment is now ready to use.

Writing your program

As mentioned in the introduction, the cockpit instrument that we are going to create is an Airspeed Indicator. Let's start by creating a folder on the Desktop and call it **airspeed_indicator**.

In this folder we need put a couple of files from the FSUIPC SDK (Software Development Kit).
You can download the SDK from http://www.schiratti.com/dowson.html.

The files you will need to copy into your **airspeed_indicator** folder are **FSUIPC_User.h** & **FSUIPC_User.lib**. You will find these files in the **UIPC_SDK_C.zip** file within the main zip file.

Also within the **airspeed_indicator** folder create 2 other folders called **bin & objects**.

At this point you would normally copy the **alleg40.dll** file from the **C:\Program Files\Dev-Cpp\Dll** directory into the **bin** directory that you have just created. However there is a problem with alleg40.dll in this package that causes a windows 'Data Execution Prevention' message to be displayed. Therefore I have included an alleg40.dll file with this tutorial (see **tutorial_files** directory) that you should copy to your **bin** directory. It is actually alleg43.dll renamed to alleg40.dll but seems to work fine.

Now for some graphics

I'm all for making things as easy as possible so the first thing we need to do is get hold of a suitable image of an Airspeed Indicator. After 2 minutes of looking on Google Images I found this one.

I'm assuming you can use a graphics editing program as to include how to edit pictures would make this tutorial huge. You can download a great free photo editing program at http://www.gimp.org/windows.

Basically you are looking to turn the above image into something like this:-

Notice the pink colour around the pointer. This is what Allegro uses as a mask. i.e. it will not be shown on the screen so you will only see the pointer. This pink colour has to have the following colour properties: Red = 255, Green = 0, Blue = 255.

Save images as .bmp files. I specify a 24 Bit Colour Depth.

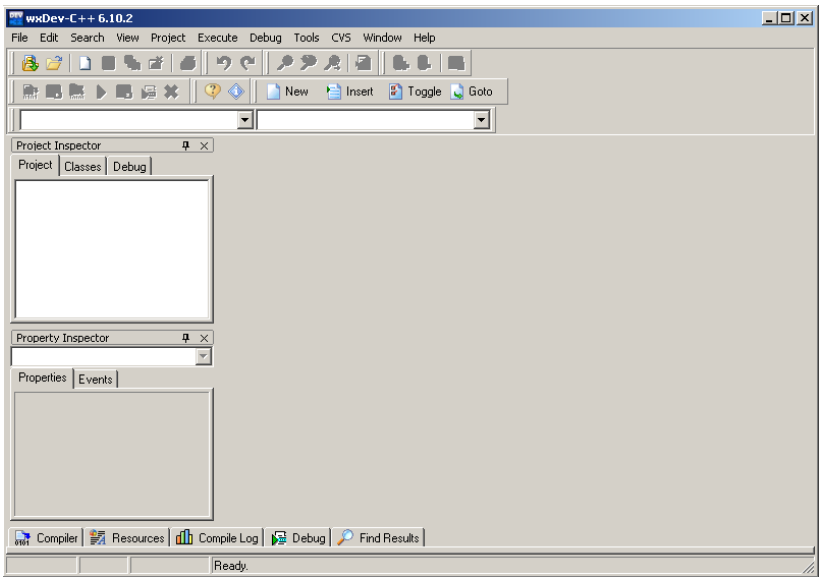
For this tutorial, use the bitmaps that I have created. Copy the **background.bmp** and **pointer.bmp** files that came with this tutorial into the **bin** directory that you created earlier.

You now have everything you need to start coding.

The Dev-C++ development environment that you have installed has its own editor.

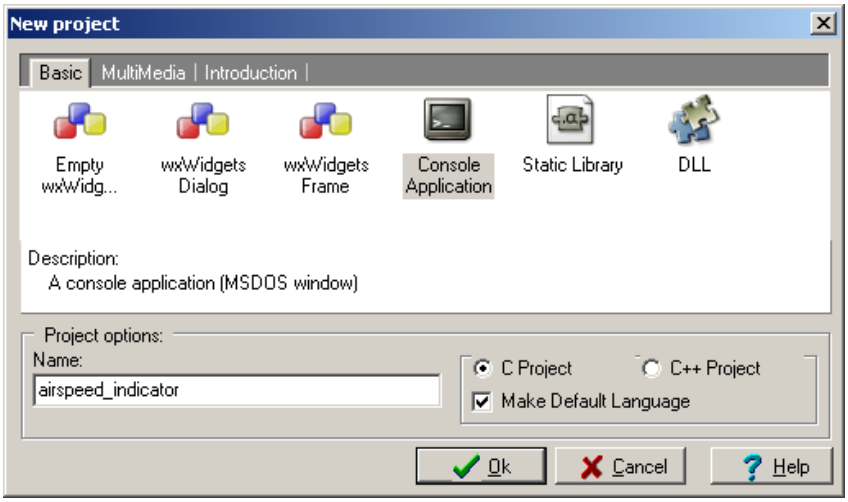
If wxDevc++ is not already running Click:-
Start – All Programs – wx-Devcpp – wx-Devcpp

You should now see the following window:-

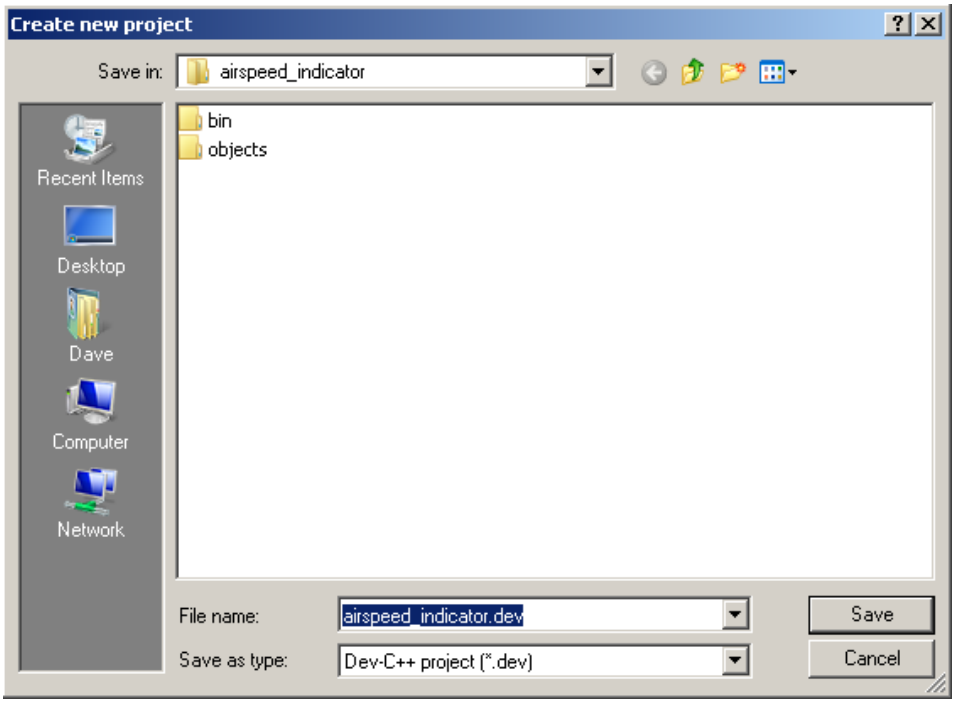


Click on **File – New – Project**

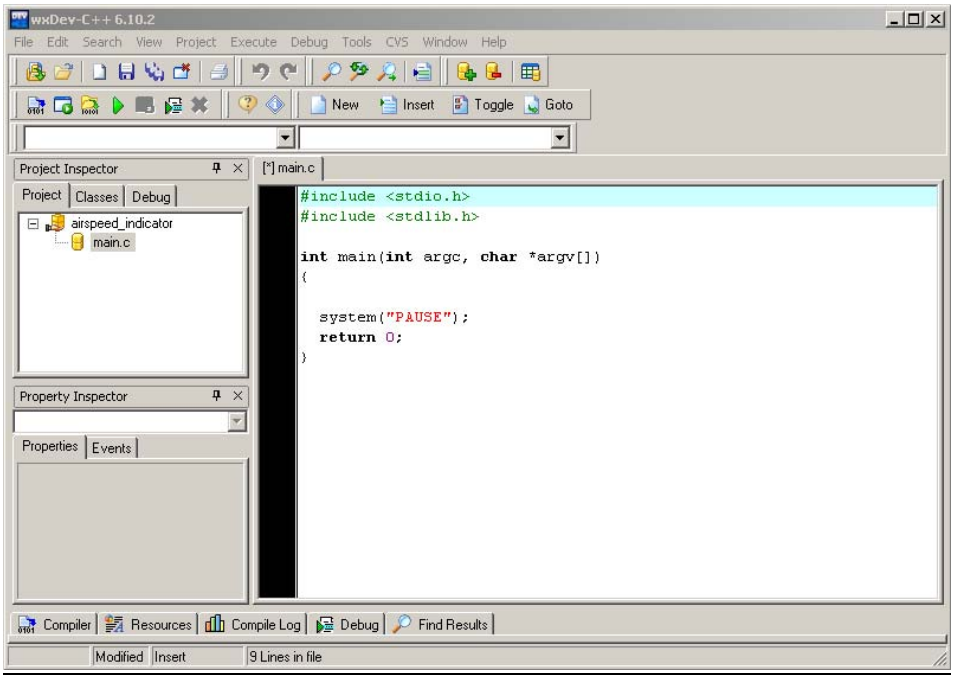
Select **Console Application**. Type **airspeed_indicator** as the project name and tick the box that says **C Project** then click **OK**.



Navigate to the airspeed_indicator folder that you created earlier and save your project as **airspeed_indicator.dev**

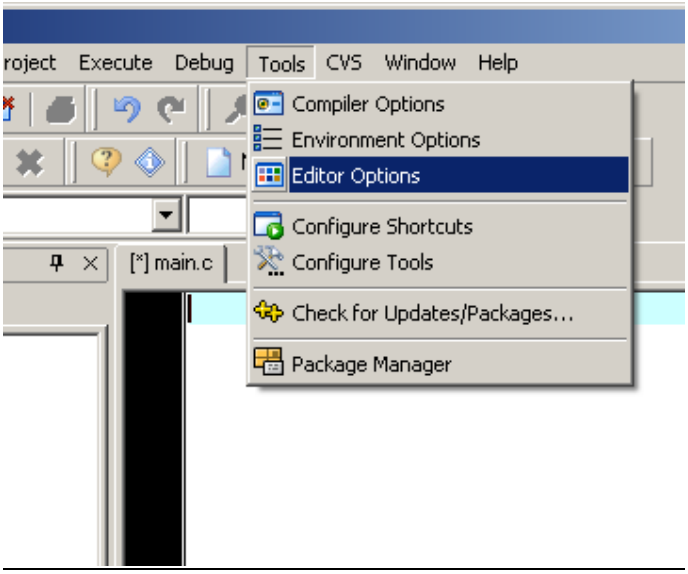


You will see that it has inserted a file called main.c under airspeed_indicator within the Project Tab and also started writing the program for you.

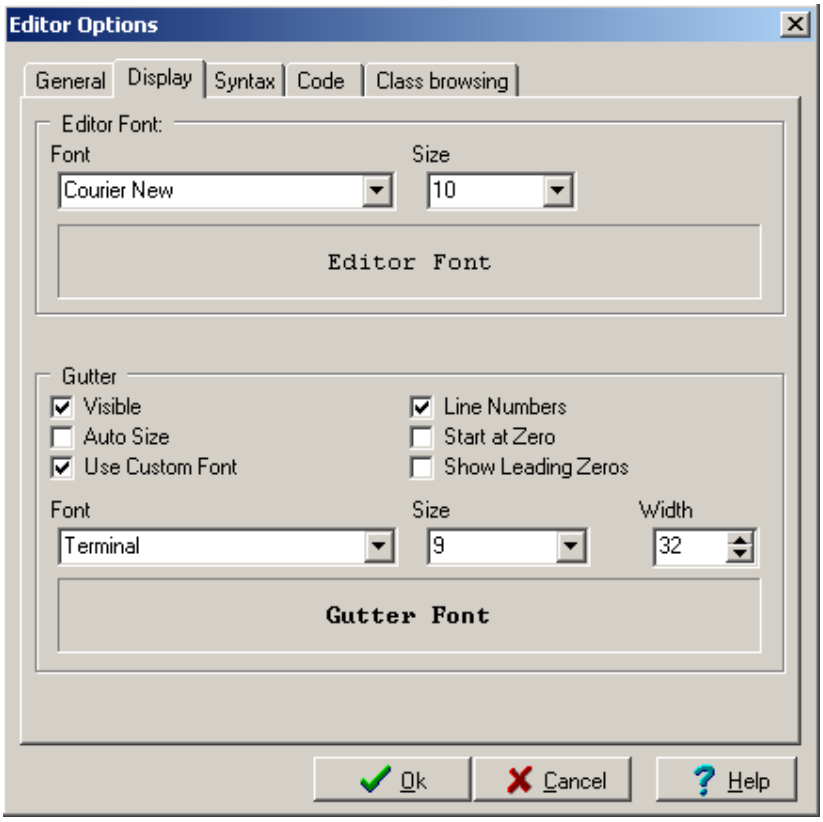


Delete the lines of code that are there.

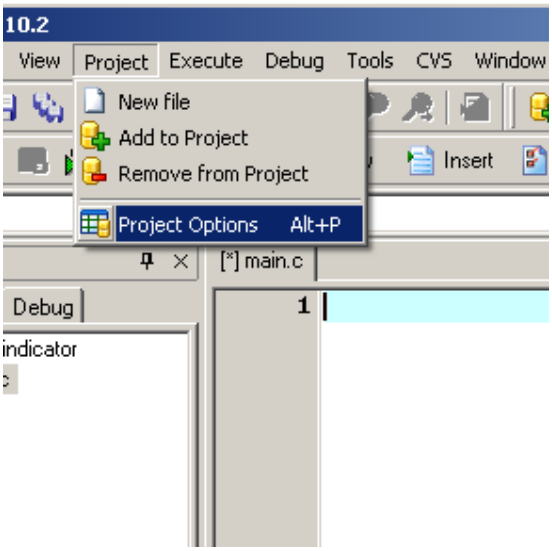
Click **Tools** – **Editor Options**.



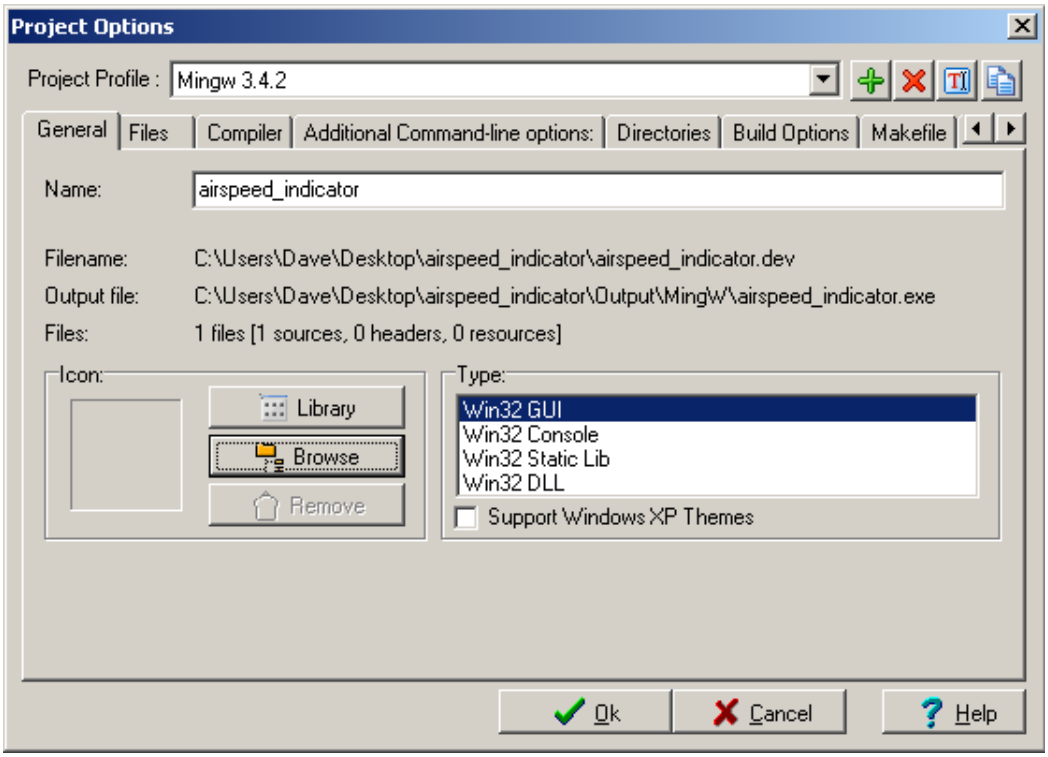
Select the **Display** tab and tick **Line Numbers**. Then click **OK**.



Click on **Project** – **Project Options**.

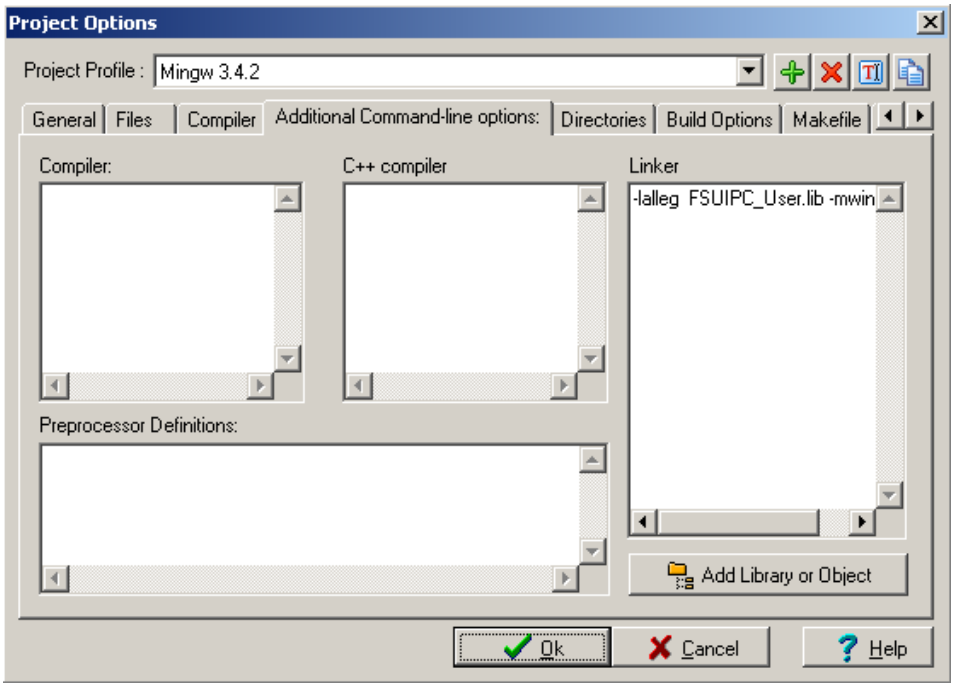


In the **General** tab, select **Win32 GUI**.

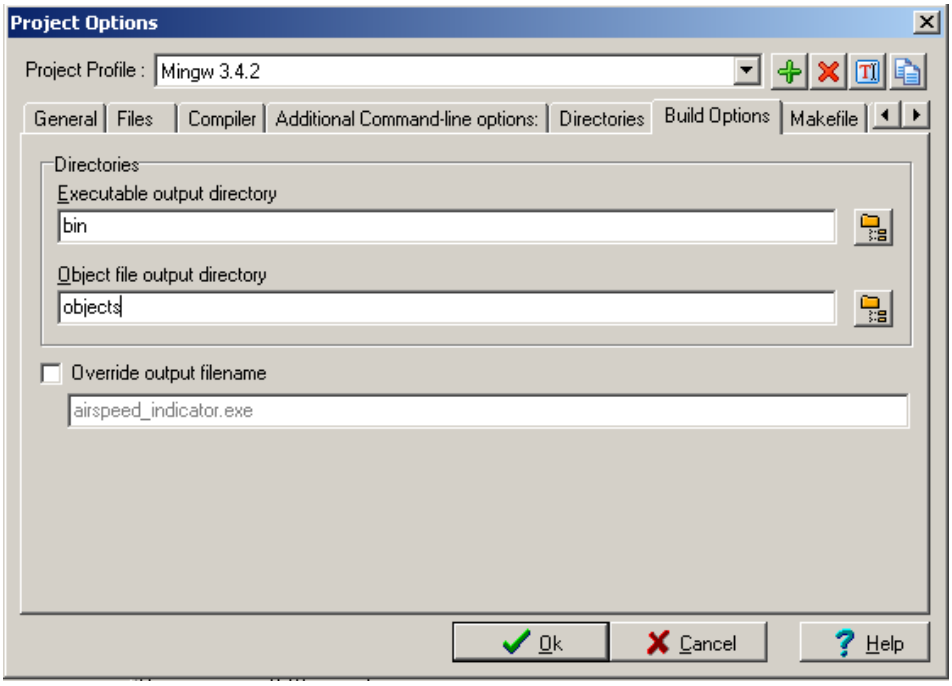


In the **Additional Command line options** tab, in the Linker box enter :-

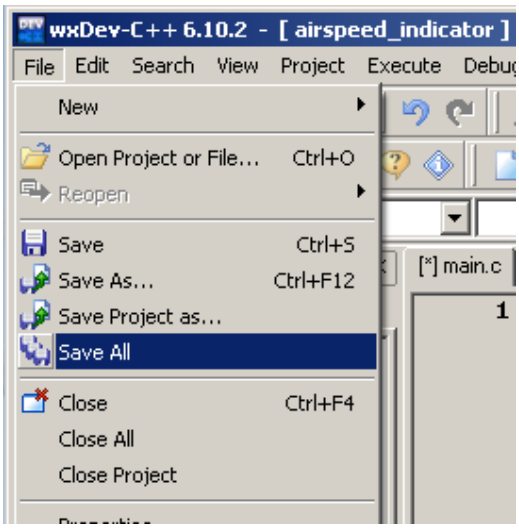
-lalleg FSUIPC_User.lib -mwindows
This tells it to use the **allegro** and **FSUIPC** libraries and that it is a windows program.



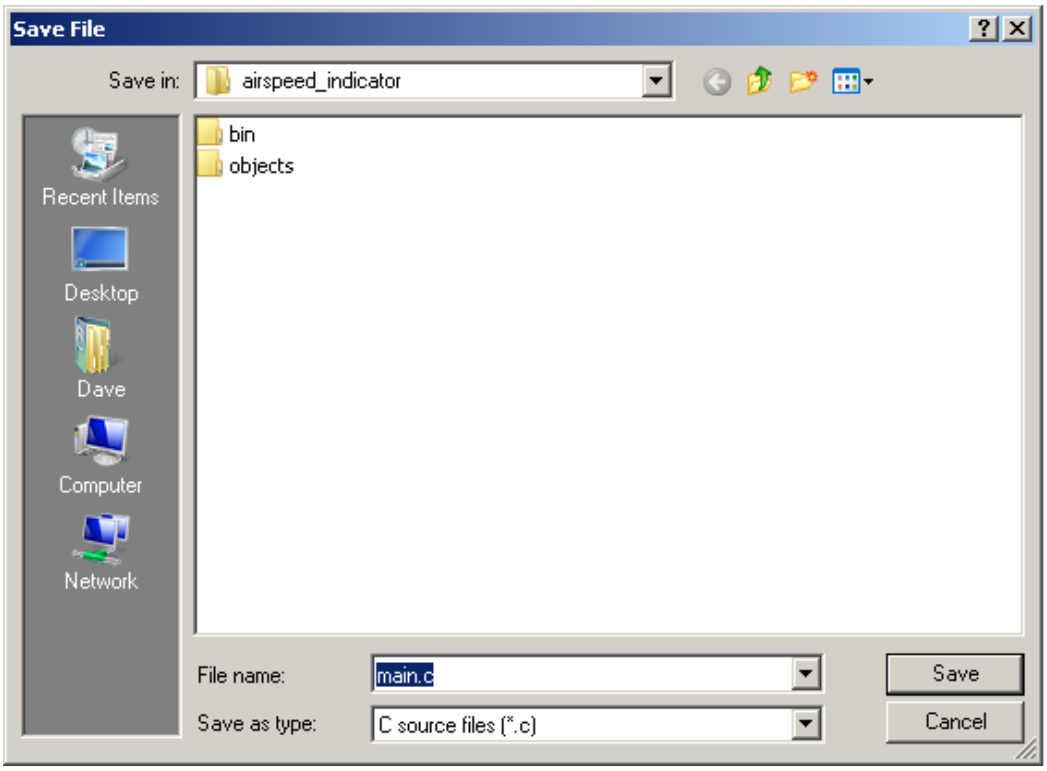
In the **Build Options** tab change **Executable output directory** to **bin** and **Object file output directory** to **objects**. Then click **OK**.



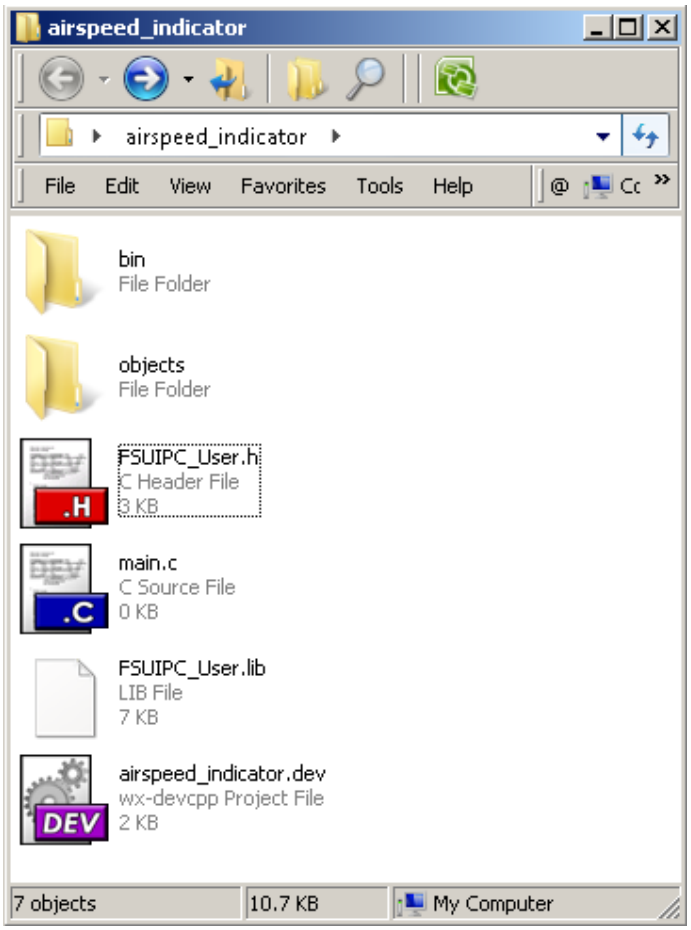
Click **File – Save All**



It will ask where you want to save main.c. Ensure it has defaulted to your **airspeed_indicator** folder and press **Save**.



Your **airspeed_indicator** directory should now look something like this.

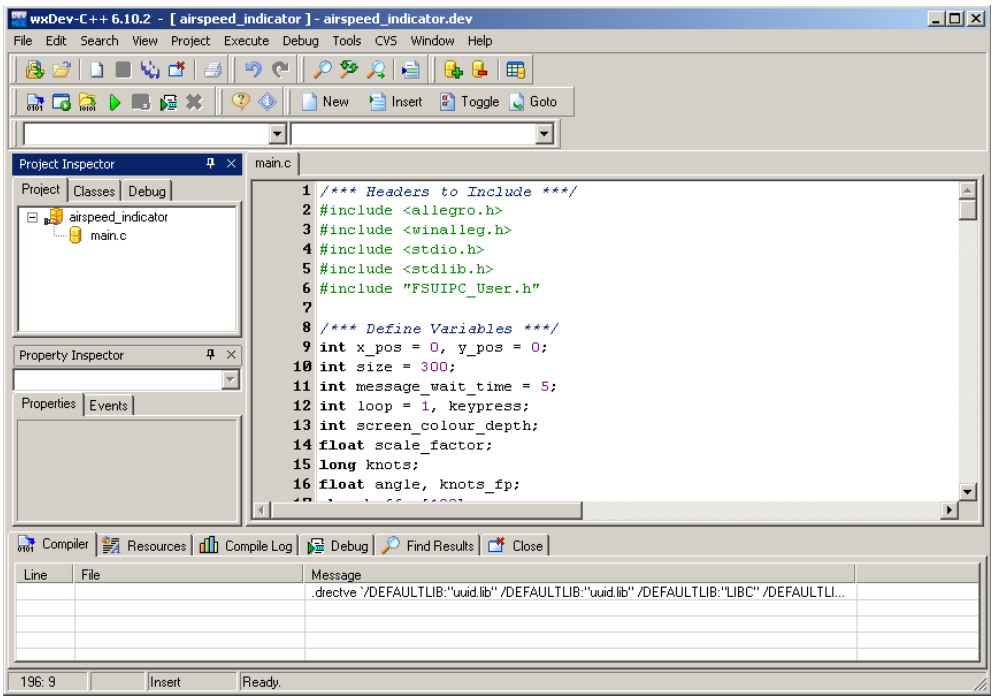


Exit wx-DevC++

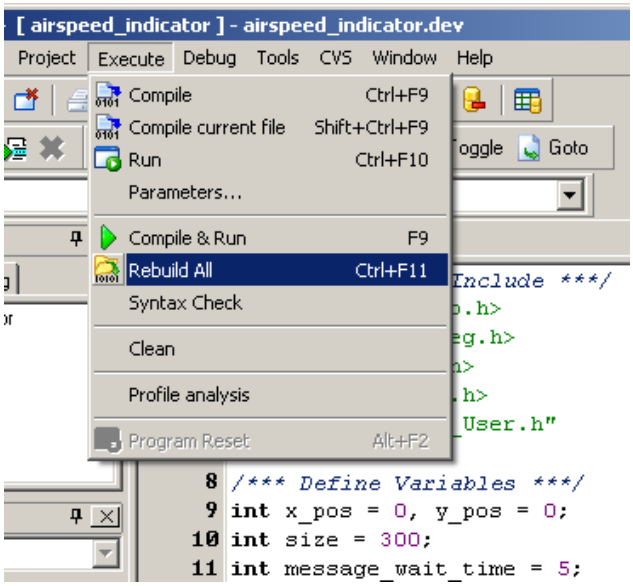
To save time typing in the program, copy the **main.c** file that accompanies this tutorial into your **airspeed_indicator** folder (overwriting the one that is already there).

To run wx-DevC++ and automatically load your project, Double click on the **airspeed_indicator.dev** file within the **airspeed_indicator** directory.

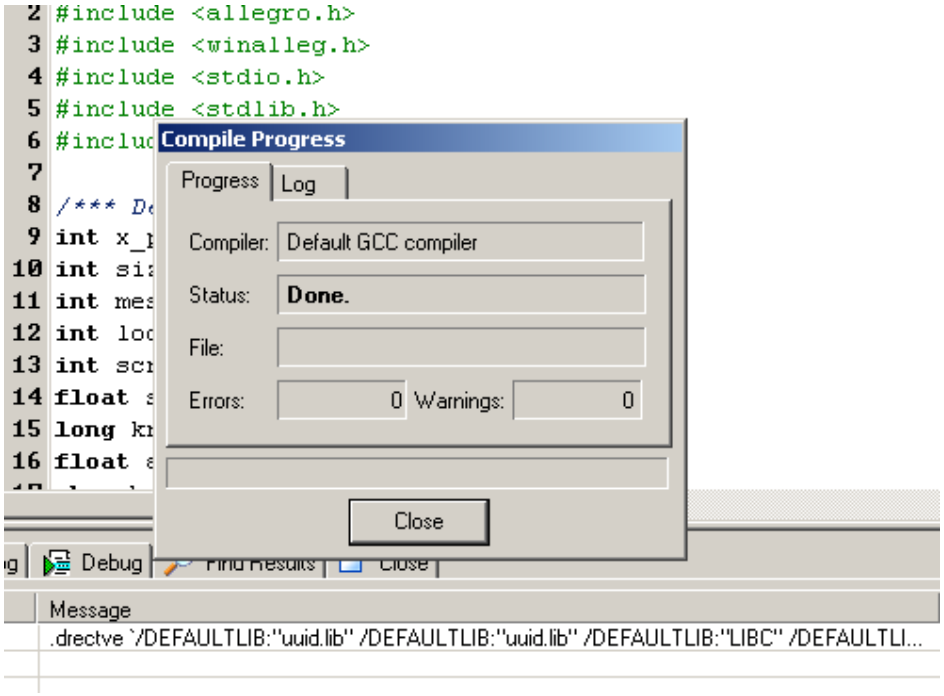
You should now see the program in the editor. If not then click on main.c in the Project Inspector window.



Click **Execute** – **Rebuild All**



The program should Compile, Link and report 0 Errors & 0 Warnings.

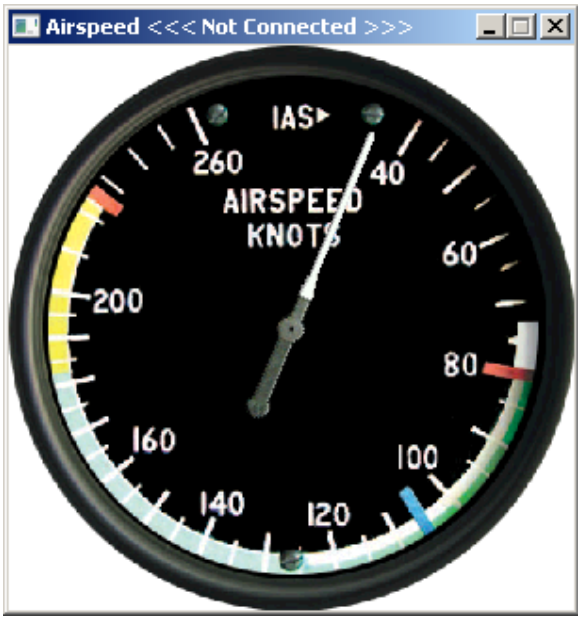


If you do get any warnings or errors then you have probably done something wrong and will need to check what you have done against this tutorial.

Note you do get a .drectve message but this is nothing to worry about.

If you look in the **bin** folder within your **airspeed_indicator** directory you should see a file called **airspeed_indicator.exe**

Double click on this file and the Airspeed Indicator program should run.



Drag the gauge to the required location on the screen. Use the '**i**' and '**d**' keys to increase or decrease the size.

Notice that it says **Not Connected** on the top bar. Assuming that you have installed and are running **FSUIPC** / **WideFS** then when you run FSX this should change to **Connected**. The pointer will start moving when speed is greater than 35 Knots.

Exit the airspeed_indicator program by clicking on the cross in the top right hand corner.

Notice that you now have a **airspeed_indicator.ini** file in your **bin** directory. When you next run **airspeed_indicator.exe** it will read this .ini file and the program will appear in the same location and be the same size as when you last exited it.

Useful reading material

If you are not familiar with the C programming language, the Allegro programming library and FSUIPC / WideFS then I suggest that you read their documentation.

There are loads of good free guides to the C programming language on the net.
A particularly good one is at :-
http://www.oucs.ox.ac.uk/documentation/userguides/c/l922.pdf

The Allegro documentation is here :-
C:\Program Files\Dev-Cpp\Docs\Allegro\allegro.html

FSUIPC / WideFS Documentation is included with the programs and programming instructions and examples are to be found in the FSUIPC SDK :-
http://www.schiratti.com/dowson.html

Understanding how the program works

```
#include <allegro.h>
#include <winalleg.h>
#include <stdio.h>
#include <stdlib.h>
#include "FSUIPC_User.h"
```

The above section defines which header files (.h files) should be included. Generally there are two types of header files. System header files, which declare the interfaces to parts of the operating system and are included in your program to supply the definitions and declarations that you need to invoke system calls and libraries. Other header files contain declarations for interfaces between the source files of your program.

stdio.h and stdlib.h should usually be included in a C program. allegro.h and winalleg.h are required for the allegro games programming features and FSUIPC_User.h is required for the commands that interrogate the simulator via FSUIPC.

```
int x_pos = 0, y_pos = 0;
int size = 300;
int message_wait_time = 5;
int loop = 1, keypress;
int screen_colour_depth;
float scale_factor;
long knots;
float angle, knots_fp;
char buffer[100];
DWORD dwResult;
HDC hdc;
FILE *inifile;
BITMAP *scrat, *background, *pointer;
PALETTE palette;
```

The above section defines the variables. int , long & float specifies which variables are defined as integers , long and floating point.

DWORD is a 32 bit unsigned integer and is required when reading and writing values via FSUIPC.

HDC (Handle to Device Context) is a special variable requires when using the function GetDeviceCaps() to read the screen colour bit depth.

FILE is used to define a pointer to a file. In this case ‘inifile’ used when loading or saving the .ini file.

BITMAP & PALETTE define variables required by Allegro when loading / storing the graphics images.

```
int initialise_window( void );
int connect_to_sim( void );
int update_display( void );
int create_bitmaps( void );
int get_knots( void );
int load_ini_file( void );
int save_ini_file( void );
int act_on_keypress( void );
int close_program( void );
void exit_hook (void );
```

I have separated out sections of the code into a number of functions. This makes the program neater and easier to understand. The above section defines variable types that are passed into these functions and also the variable type that is returned by the function. Void indicates that nothing is to be passed into these functions when they are called and all return an integer except the exit_hook() function.

```
int main( void )
{
    // Load window position and window size from .ini file
    load_ini_file();

    // Create window, Start Allegro, Display window & Get / Set colour depth
    initialise_window();

    //Loads the graphic files ( background & pointer.bmp) . Displays Error if unsuccessful
    if( create_bitmaps() == -1 )
    {
        textprintf_centre(screen, font, size / 2, size * 0.15, 65535,  "***
Error ***" );
        textprintf_centre(screen, font, size / 2, size * 0.40, 65535,  "No
Graphics files" );
        textprintf_centre(screen, font, size / 2, size * 0.65, 65535,
"Press Key to Exit" );

        // Wait for key press
        readkey();

        // Terminate Allegro
        close_program();

        // Return a value of  -1 when exiting
        return( -1 );
    }

    // Draws the background image and the pointer
    update_display();

    //Establish a connection to the simulator
    connect_to_sim();

    //Repeat  until loop is not equal to 1
    while( loop == 1 )
    {
        //Read speed from Simulator
        get_knots();

        //Loop while a key is pressed
        while( keypressed() )
        {
            //Process key press
            act_on_keypress();

            //Draw / redraw instrument
```

```
            update_display();
        }
        //Clears the keyboard buffer to prevent buffered keystrokes
        clear_keybuf();

        //Draw / redraw instrument
        update_display();
    }
    //Save window position & window size to a .ini file
    save_ini_file();

    //Terminate Allegro and disconnect FSUIPC
    close_program();

    //Return a value of 0 when exiting
    return (0);
} END_OF_MAIN()   //This is required at the end of the main function for all Allegro programs
```

Detailed descriptions of each function:-

//Function to Create window, Start Allegro & Get / Set colour depth

```
int initialise_window( void )
{
    //These two lines get the display colour depth
    hdc = GetDC(NULL);
    screen_colour_depth = GetDeviceCaps (hdc, BITSPIXEL );

    //Initialise Allegro & install Allegro keyboard interrupt handler
    if (allegro_init() != 0) return 1;
    install_keyboard();

    //Forces colour depth to be same as display
    set_color_depth(screen_colour_depth);

    //The next few lines create an initial window of 800x600
    if (set_gfx_mode(GFX_AUTODETECT_WINDOWED, 800, 600, 0, 0) != 0) {
            if (set_gfx_mode(GFX_SAFE, 800, 600, 0, 0) != 0) {
                set_gfx_mode(GFX_TEXT, 0, 0, 0, 0);
                allegro_message("Unable to set any graphic mode\n%s\n",
allegro_error);
                return 1;
            }
    }
```

//Resize and position window based on values read from the .ini file. 6 is added to width and 25 to height as you have to take into account the size of the windows borders.

```
      SetWindowPos(win_get_window(), 0, x_pos, y_pos, size+6, size+25,
SWP_ASYNCWINDOWPOS);

    //Ensures that the program will carry on running in the background
    set_display_switch_mode(SWITCH_BACKGROUND);

    //Factor to multiply by to ensure graphics are scaled correctly whatever the window size is set to
    scale_factor = ((float)size / 419);

    //Specifies that the function exit_hook() is run when the window close is clicked on
    set_window_close_hook(exit_hook);

    //Returns a value of zero ( This value is not used )
    return( 0 );
}
```

//Function to establish a connection to the Sim via FSUIPC / WideFS

```
int connect_to_sim( void )
{
    //Set Indicator  to start  & update display
    knots_fp = 35.0;
    update_display();

    //Loop until connection is established or loop is not equal to 1
    while (!FSUIPC_Open(SIM_ANY, &dwResult) && loop == 1 )
    {
            //Change window title to show 'not connected' status
            set_window_title( "Airspeed <<< Not Connected >>>" );

            //Wait 100milliseconds. Reduces Network and CPU load
            Sleep( 100 );

            //Loop while a key is pressed
            while( keypressed() )
            {
                //Process keypress and update display
                act_on_keypress();
```

```
                update_display();
            }
            //Clears the keyboard buffer to prevent buffered keystrokes
            clear_keybuf();
        }
        //Change window title to show 'connected' status
        set_window_title( "Airspeed >>>   Connected   <<<" );

        //Send a 'connected' message to the Simulator screen
        if (FSUIPC_Write(0x3380, 29, "Airspeed Indicator Connected\0",
&dwResult)) FSUIPC_Process(&dwResult);
        if (FSUIPC_Write(0x32FA, 2, &message_wait_time, &dwResult))
FSUIPC_Process(&dwResult);

        //Return a value of 0 when exiting function
        return( 0 );
}

//Function to update the display
int update_display( void )
{
    //Ensure speed indication cannot be less than 35 knots or greater than 262 knots
    if( knots_fp < 35 ) knots_fp = 35;
    else if( knots_fp > 262 ) knots_fp = 262;

    //Calculate angle of the pointer using a 4th order polynomial to compensate for the non linear scale
    /*
     * Ok, this sounds really complicated but it's actually quite easy.
     * I created a spreadsheet with one column containing the knots values ( 0 – 260 ) in 5 knot
     * increments. Then compiled and ran this program with manually entered angles to determine
     * the angle for each speed value in the spreadsheet. I entered this angle value in the second
     * column. I then plotted knots vs. angle. I clicked on the plotted  line to highlight it then right
     * clicked and selected 'Add Trendline'. I selected  Polynomial, Order 4 and ticked
     *'Display Equation on chart'.
     * I then used this equation to create another column of calculated values and tweaked the poly
     * values to get the best match  ( spreadsheet attached ).
     * Note: Allegro uses a value of 0-255 for the angle to give 0-360 degrees.
     */
    angle = (6.06e-8*pow(knots_fp,4)) - (3.98e-5*pow(knots_fp,3)) + (7.05e-
3*pow(knots_fp,2)) + (0.78*knots_fp) - 18.28;

    //This copies the whole of the background image into the scaled scrat bitmap
    stretch_blit( background, scrat, 0,0,419,419,0,0,size,size);

    //This copies the pointer bitmap / sprite into the scaled scrat bitmap. It is sized by scale_factor.
    pivot_scaled_sprite(scrat, pointer, size / 2, (size / 2), 17, 167,
ftofix(angle) , ftofix(scale_factor));

    //The scrat bitmap is then copied to the screen ( window ).  This is a double buffering technique
    // that prevents you seeing the redraws and gives smoother graphics.
    blit(scrat, screen, 0, 0, 0, 0, size, size);

    //Small delay added to reduce CPU load
    Sleep(10);

    // Return a value of 0 when exiting function
    return( 0 );
}
```

```
// Function to create bitmaps
int create_bitmaps( void )
{
     //Create a bitmap to be used to write the graphics to before copying to screen ( double buffering )
     scrat = create_bitmap(size, size);

     //Create Background and Pointer Bitmaps from .bmp file images. If unsuccessful, return -1
     if( (background = load_bitmap( "background.bmp", palette )) == NULL )
return( -1 );
     if( (pointer = load_bitmap( "pointer.bmp", palette )) == NULL ) return(
-1 );
     // Return a value of 0 when exiting function
     return( 0 );
}

// Function to read airspeed from the simulator program
int get_knots( void )
{
       // Get airspeed from Sim via FSUIPC / WideFS ( value read is Knots * 128 )
       // If at this point it fails to read from the sim it runs the connect_to_sim() function
       if(!FSUIPC_Read(0x02BC, 4, &knots, &dwResult) ||
!FSUIPC_Process(&dwResult)) { FSUIPC_Close(); connect_to_sim(); }

       // Stores knots as a floating point, this gives a smoother pointer movement than integer value
       knots_fp = (float)knots / 128;

       // Return a value of 0 when exiting function
       return( 0 );
}

// Function to load window position and window size if .ini file is available
int load_ini_file( void )
{
     // Open .ini file if it exists
     if( (inifile = fopen( "airspeed_indicator.ini", "r" )) != NULL )
     {
          // If not at end of file then read 1st line of .ini file as a string, convert it to integer and
          // store it as x_pos
          if( !feof( inifile ))
          {
              fgets( buffer, 99, inifile );
              x_pos = atoi( buffer );
          }

          // If not at end of file then read 2nd line of .ini file as a string, convert it to integer and
          // store it as y_pos
          if( !feof( inifile ))
          {
              fgets( buffer, 99, inifile );
              y_pos = atoi( buffer );
          }

          // If not at end of file then read 3rd  line of .ini file as a string, convert it to integer and
          // store it as size
          if( !feof( inifile ))
          {
              fgets( buffer, 99, inifile );
              size = atoi( buffer );
          }
          // Close the .ini file
          fclose( inifile );
     }

     // Return a value of 0 when exiting function
     return( 0 );
```

```
}

// Function to save Window position and size to a .ini file
int save_ini_file( void )
{
     // Get window properties to determine position and size ( stored in structure 'rect' )
     RECT rect;
     if (GetWindowRect(win_get_window(), &rect))
     {
          // Create / Overwrite .ini file
          if( (inifile = fopen( "airspeed_indicator.ini", "w" )) != NULL )
          {
               //Write X and Y coordinates of the window and its size to the .ini file
               fprintf( inifile, "%li\n",rect.left);
               fprintf( inifile, "%li\n",rect.top);
               fprintf( inifile, "%d\n",size );

               // Close the .ini file
               fclose( inifile );
          }
     }
     // Return a value of 0 when exiting function
     return( 0 );
}

// Function to process keyboard key presses
int act_on_keypress( void )
{

     // Get window properties to determine position ( stored in structure 'rect' )
     RECT rect;
     if (GetWindowRect(win_get_window(), &rect))
     {
          // Save position in x_pos & y_pos variables
          x_pos = (int)rect.left;
          y_pos = (int)rect.top;
     }

     // Read integer value of  key pressed
     // See program included with this tutorial that displays key codes
     keypress = readkey();

     //Run code for whichever key pressed
     switch (keypress)
     {
          // 'i' key pressed ( increase windows size )
          case 2409:

               // Only increase windows size if size less than 600
               if ( size < 600 ) size+=1;

               // Recreate scrat bitmap with new size
               destroy_bitmap( scrat );
               scrat = create_bitmap(size, size);

               //Scale window to new size
               SetWindowPos(win_get_window(), 0, x_pos, y_pos, size+6, size+25,
SWP_NOZORDER);

               //Recalculate scale factor
               scale_factor = ((float)size / 419);

               // Causes the switch statement to conclude
               break;
```

```
            // 'd' key pressed ( decrease windows size )
            case 1124:

                // Only decrease windows size if size greater than 142
                if( size > 142 ) size-=1;

                // Recreate scrat bitmap with new size
                destroy_bitmap( scrat );
                scrat = create_bitmap(size, size);

                //Scale window to new size
                SetWindowPos(win_get_window(), 0, x_pos, y_pos, size+6, size+25,
SWP_NOZORDER);

                //Recalculate scale factor
                scale_factor = ((float)size / 419);

                // Causes the switch statement to conclude
                break;
        }
        // Return a value of 0 when exiting function
        return( 0 );
}

// Function to tidy up before the program closes.
int close_program( void )
{
        //Shuts down the FSUIPC connection
        FSUIPC_Close();

        //Shuts down Allegro
        allegro_exit();

        // Return a value of 0 when exiting function
        return( 0 );
}

// Function called when window is closed
void exit_hook( void )
{
        // Setting loop = 0 causes it to exit from the above loops in main() and connect_to_sim()
        loop = 0;
}
```

I hope that this tutorial has given you a basic understanding of how to write your own Cockpit Instrument and also given you the confidence to build on this knowledge and have a go at writing your own.

Like I said in the Introduction, I am not a qualified programmer and when I write a program I have a 'programming in C' book on one side and the 'Allegro programming guide' on the other side and I keep trying things until I get the results that I want. This is a great way of learning.

Cheers

Dave
www.learjet45chimera.co.uk