Program of the Week PW7

This program will follow the format of PW#6. If these steps are not clear then refer back to PW#6 for illustrations of details.

An existing program will be loaded into a CodeBlocks Empty Project. First thing is to create the Empty Project. Then download the program code and put into the project. The code will then be edited, compiled and run for demonstration. This is similar process as for the samples distributed with wxWidgets source package.

1. Start CodeBlocks
2. Create a new project
3. Select wxWidgets version – wxWidgets 3.0.x
4. Project title – this can be title desired for this project like wxTimerDemo
5. Author – my name i.e. HW
6. Preferred GUI builder: none
   Application type: Frame
7. wxWidget's location
   C:\wxWidgets-3.1.0
8. Create "Release" Configuration
9. Miscellaneous Settings – Select box for Create Empty Project
10. Ignore the Matching Release configuration Warning

Now there is a project called wxTimerDemo with no source or header files in it. The only file in the folder is the Code::Blocks project file – wxTimerDemo.cbp.

Next a source file will be created and the source code loaded from the class web site

Open the class web site: http://web.eng.fiu.edu/watsonh/eel3370/index.html

Select the link for the source code for the RenderTimer
http://web.eng.fiu.edu/watsonh/eel3370/ProgOfWeek/RenderTimer.cpp

11. Once the web page is opened, highlight all the code – Ctrl-A, then copy the code to the Clipboard - Ctrl-C

12. Back to the CodeBlocks TimerDemo project. Select File->New->Empty File

13. A dialog appears - "Do you want to add this new file in the active project?" - YES

14. Save File Dialog appears – enter new file name: main.cpp

15. in the main.cpp edit screen: Paste the Clipboard contents – Ctrl-V

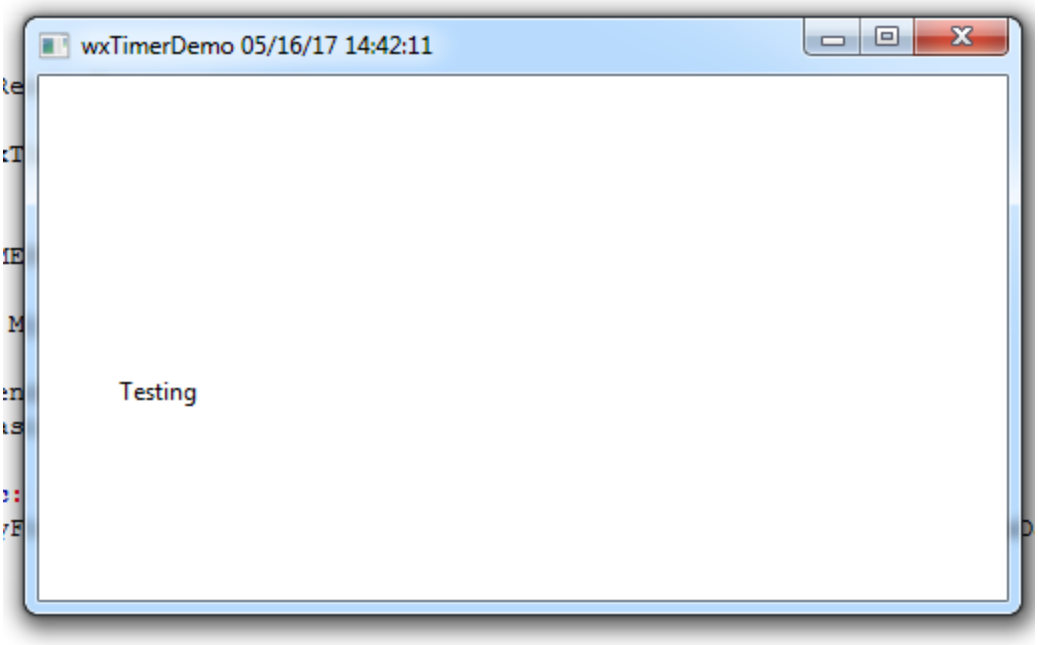
16. Set the Linker settings for the wxWidgets 3.1.0 library
    1. Right click project name (in the Management frame)
    2. Select Linker settings
    3. Edit libwxmsw30u.a – change to libwxmsw31u.a for wxWidgets 3.1
       remember Step-by-Step Install Instructions Slides 31-34 – every project!

17. Add the Frame title including the Date and Time.

```
64  class MyFrame : public wxFrame
65  {
66      RenderTimer* timer;
67      BasicDrawPane* drawPane;
68
69  public:
70      MyFrame() : wxFrame((wxFrame *)NULL, -1,  wxT("wxTimerDemo " + wxDateTime::Now().Format("%c"))
71                          , wxPoint(50,50), wxSize(500,300))
72      {
```

18. Build and Run the project

Capture the screen with the application running – as below:



19. Turn in answers to the following study questions plus a copy of the screen as above.

Study Questions: study the source code and answer the following questions:

Question 1. Which of the 3 different ways is the timer used in this program

Question 2. What is the period for the timer used?

Question 3. Which object and function is called with timer repeated periodic notification?

Question 4. What event causes the text to be moved?

- Documentation for all wxWidget classes
  http://docs.wxwidgets.org/3.0/annotated.html
- documentation for the Timer widget
  http://docs.wxwidgets.org/3.0/classwx_timer.html
- documentation for the wxPanel widget
  http://docs.wxwidgets.org/3.0/classwx_panel.html

wxTimer:

Using wxTimer

The `wxTimer` class lets your application receive periodic notification, either as a "single shot" or repeatedly.

You can choose how your code will be notified. If you prefer to use a virtual function, derive a classfrom `wxTimer` and override the `Notify` function. If you prefer to receive a `wxTimerEvent`event, pass a `wxEvtHandler` pointer to the timer object (in the constructor or using `SetOwner`), and use `EVT_TIMER(id, func)` to connect the timer to an event handler function.

Optionally, you can pass an identifier that you passed to the constructor or `SetOwner` to uniquely identify the timer object and then pass that identifier to `EVT_TIMER`. This technique is useful if you have several timer objects to handle.

Start the timer by calling `Start`, passing a time interval in milliseconds and `wxTIMER_ONE_SHOT` if only a single notification is required. Calling `Stop` stops the timer, and `IsRunning` can be used to determine whether the timer is running. From : http://flylib.com/books/en/3.138.1.146/1/

How to paint the contents of wxPanel using the timer:

The sequence for timer notifications is as follows:

- In the notification function (Question 3.), the inherited wxWindow::Refresh() is called.
- Refresh() creates a wxPaintEvent which calls our application BasicDrawPane::paintEvent.
- The function BasicDrawPane::paintEvent():
  - creates a 'Device Context' for 'this' window
  - calls BasicDrawPane::render()
- The function BasicDrawPane::render():
  - creates new x and y coordinates
  - clears the wxPanel Device Context
  - draws the word “Testing” on the wxPanel Device Context

References:

The drawing part of this application is derived from a 'Minimal Drawing Sample' at:
http://wiki.wxwidgets.org/WxDC

A good guideline of the overall process:
http://wiki.wxwidgets.org/Drawing_on_a_panel_with_a_DC

With timer notification, the Refresh() member function from wxWindow (which wxPanel inherits from) causes this window, and all of its children recursively to be repainted – including the panel.
http://docs.wxwidgets.org/3.0/classwx_panel.html
http://docs.wxwidgets.org/3.0/classwx_window.html

When the window is 'Refreshed' , it causes the window to be repainted. A paint event is sent when a window's contents needs to be repainted.
http://docs.wxwidgets.org/3.0/classwx_paint_event.html#details

the wxPaintEvent calls BasicDrawPane::paintEvent(wxPaintEvent& evt). That function creates a wxPaintDC from 'this' wxPanel.
http://docs.wxwidgets.org/3.0/classwx_paint_d_c.html