Programming Windows

Terry Marris Feb 2013

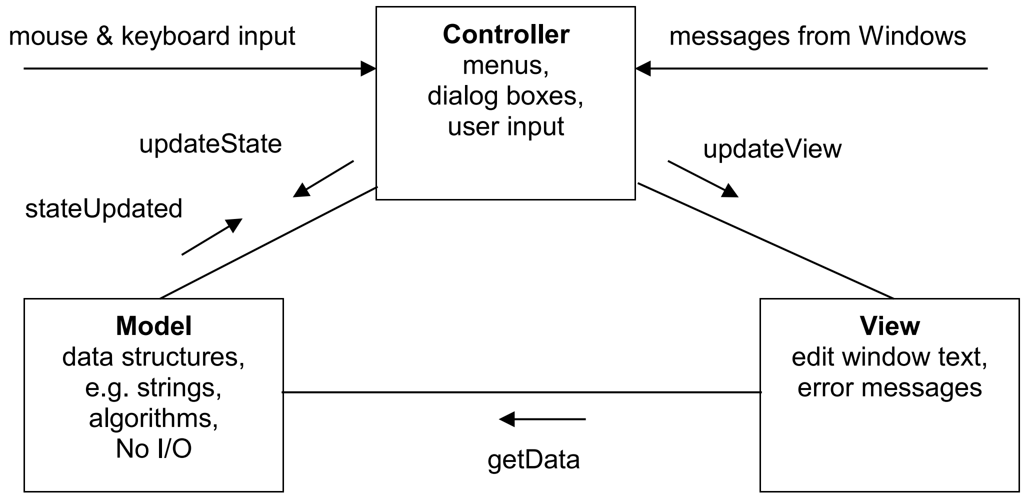
13 Model View Controller

Model-View-Controller is a design pattern that separates the user interface from the data structures and algorithms that model the application.

Model - data structures and algorithms with no reference whatsoever to input-output. Responds to requests for information (usually from the View) about its state (i.e. variables), and responds to instructions (usually from the Controller) to update the state

View - manages the user's view of the data on the screen or on the printer

Controller - interprets mouse and keyboard inputs. Includes menus and dialog boxes. Instructs the model or view to change as required.

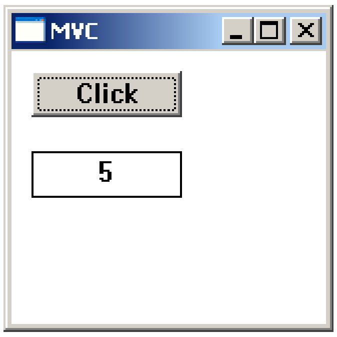


In a text editor, the *View* is informed of each change to the text so that it can update its display. The *Controller* is responsible for notifying the *View* of any changes because it interprets user's requests. It tells the *View* that something has changed, and the *View* then requests the current state from the *Model*. The *Model* is a passive holder of the string data accessed by the *View* and manipulated the *Controller*. The *Model* adds, removes or replaces substrings on demand from the *Controller*, and returns appropriate substrings on demand from the *View*. The *Model* is totally unaware of the existence of both the *View* and the *Controller*.

Ok. That's the theory. Can it be done in practice?

13.1 Model

The next program implements a simple counter. You click the button and the counter advances by 1. The value of the counter is shown in a text box. Here, the button has been clicked five times.



The model is just a simple counter.

```
/* model.c - implements a counter */

#include <windows.h>

static int counter = 0;

BOOL incrementCounter()
{
  counter++;
  return TRUE;
}


int getCounter()
{
  return counter;
}
```

First, there is no reference whatsoever to any other module, or to any input-output. Second, the variable *counter* is declared as global to the file. The intention is that other modules cannot access it except through the two functions, *incrementCounter* and *getCounter*. Third, the value returned by *incrementCounter* signals that an update has taken place.

13.2 MVC Header File

The header file contains references to the two functions in *model.c* (shown above) and to the two functions in *view.c* (shown below).

```
/* mvc.h - header file for Model-View-Controller */

#if !defined MVC_H
#define MVC_H

int newTextWindow(HWND);
int updateView();

BOOL incrementCounter();
int getCounter();

#endif
```

13.3 View

view.c presents the users view of the data from the *model*.

```
/* view.c - the user's view of the data */

#include <windows.h>
#include <stdio.h>
#include "mvc.h"

static HWND hwndResult;
static int counter;
static char strCounter[32];

int newTextWindow(HWND hwnd)
{
   hwndResult = CreateWindow("EDIT", NULL,
                   WS_CHILD | WS_VISIBLE | WS_BORDER |
                   ES_CENTER, 10, 50, 75, 23,
                   hwnd, NULL, NULL, NULL);

   counter = getCounter();
   sprintf(strCounter, "%d", counter);
   SetWindowText(hwndResult, strCounter);
   return 0;
}


int updateView()
{
   counter = getCounter();
   sprintf(strCounter, "%d", counter);
   SetWindowText(hwndResult, strCounter);
   return 0;
}
```

newTextWindow creates a text box to contain the value of *counter*. In *newTextWindow* it is expected that the first call to *getCounter* returns *counter*'s initial value, namely zero. *sprintf* converts the integer returned by *getCounter* into a string.

updateView updates the value shown in the text box by the value returned by *getCounter*.

13.4 Controller

controller.c carries with it the usual Windows overhead. A window class is created. The main window is initialised. And the message loop is set up.

```
/* controller.c - processes user input.
                  Instructs model to update its state.
                  Instructs View to refresh itself.
*/

#include <windows.h>
#include "mvc.h"

#define BTN_CLICKED 100

LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);


int WINAPI WinMain(HINSTANCE hInst, HINSTANCE hPrevInst,
                   LPSTR cmdLine, int cmdShow)
{
   static char appName[] = "MVC";
   WNDCLASS wc;
   HWND hwnd;
   MSG msg;

   wc.style = CS_HREDRAW | CS_VREDRAW;
   wc.lpfnWndProc = WndProc;
   wc.cbClsExtra = 0;
   wc.cbWndExtra = 0;
   wc.hInstance = hInst;
   wc.hIcon = NULL;
   wc.hCursor = LoadImage(NULL, IDC_ARROW, IMAGE_CURSOR,
                          0, 0, LR_SHARED);
   wc.hbrBackground = (HBRUSH)GetStockObject(WHITE_BRUSH);
   wc.lpszMenuName = NULL;
   wc.lpszClassName = appName;

   RegisterClass(&wc);

   hwnd = CreateWindow(appName, "MVC",
                       WS_OVERLAPPEDWINDOW,
                       CW_USEDEFAULT, CW_USEDEFAULT,
                       CW_USEDEFAULT, CW_USEDEFAULT,
                       NULL, NULL, hInst, NULL);

   ShowWindow(hwnd, cmdShow);
   UpdateWindow(hwnd);
   while (GetMessage(&msg, NULL, 0, 0)) {
      TranslateMessage(&msg);
      DispatchMessage(&msg);
   }
   return msg.wParam;
}


LRESULT CALLBACK WndProc(HWND hwnd, UINT msg,
                   WPARAM wParam, LPARAM lParam)
{
   switch (msg) {
   case WM_CREATE:
      CreateWindow("BUTTON", "Click",
                    WS_TABSTOP | WS_CHILD | WS_VISIBLE |
                    BS_PUSHBUTTON,
                    10, 10, 75, 23,
                    hwnd,
                    (HMENU)BTN_CLICKED,
                    (HINSTANCE)GetWindowLongPtr
                              (hwnd, GWL_HINSTANCE),
                    NULL);
      newTextWindow(hwnd);
      return 0;

   case WM_COMMAND:
      if (HIWORD(wParam) == BN_CLICKED) {
         if (LOWORD(wParam) == BTN_CLICKED) {
            if (incrementCounter())
               updateView();
         }
      }
      return 0;

   case WM_CLOSE:
      DestroyWindow(hwnd);
      return 0;

   case WM_DESTROY:
       PostQuitMessage(0);
       return 0;
   }
   return DefWindowProc(hwnd, msg, wParam, lParam);
}
```

Initially, in *WndProc,* the button is created.

```
   case WM_CREATE:
      CreateWindow("BUTTON", "Click",
                    WS_TABSTOP | WS_CHILD | WS_VISIBLE |
                    BS_PUSHBUTTON,
                    10, 10, 75, 23,
                    hwnd,
                    (HMENU)BTN_CLICKED,
                    (HINSTANCE)GetWindowLongPtr
                              (hwnd, GWL_HINSTANCE),
                    NULL);
```

And a call is made to *newTextWindow*.

```
      newTextWindow(hwnd);
```

You may remember that *newTextWindow* is defined in *view.c*.

A button click is checked for in response to the *WM_COMMAND* message.

```
   case WM_COMMAND:
      if (HIWORD(wParam) == BN_CLICKED) {
         if (LOWORD(wParam) == BTN_CLICKED) {
            if (incrementCounter())
               updateView();
         }
      }
```

A *BN_CLICKED* message is sent when a user clicks a button. *HIWORD(wParam)* specifies the notification code, *BN_CLICKED* in this case. *LOWORD(wParam)* contains the button's control identifier, *BTN_CLICKED*. This identifier is declared at the top of *controller.c*.

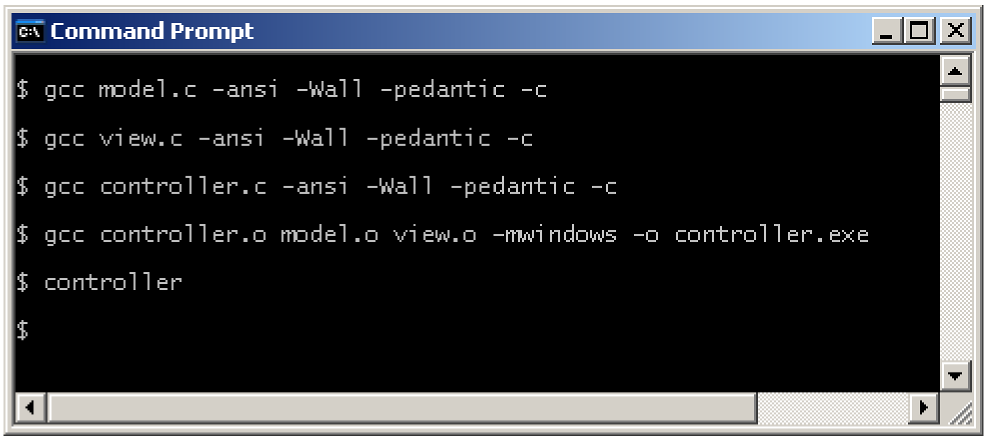
```
#define BTN_CLICKED 100
```

incrementCounter, defined in *model.c*, returns *TRUE* if *counter* has been updated. *updateView*, defined in *view.c*, retrieves the updated value of *counter*.

And there you are. Perhaps an improvement would be to have *model* issue a message when its state has been changed, and for *controller* to process this message.

13.5 Compiling and Linking

Just to remind you, with the gnu compiler collection C compiler at the MS-DOS prompt, the individual modules are compiled separately before linking and running.



```
$ gcc model.c -ansi -Wall -pedantic -c
$ gcc view.c -ansi -Wall -pedantic -c
$ gcc controller.c -ansi -Wall -pedantic -c
$ gcc controller.o model.o view.o -mwindows -o controller.exe
$ controller
$
```

Bibliography

BURBECK S, How to Use Model-View-Controller, *http://st-www.cs.illinois.edu/users/smarch/st-docs/mvc.html*, accessed Feb 2013

HOPKINS T, HORAN B, *Smalltalk: An Introduction to Application Development Using Visual Works*, Prentice Hall 1995