Lab 5.5: Adding a Time Pane to the Status Bar

Objectives

At the end of this lab, you will be able to:

- Add a pane to a status bar.
- Provide a handler to update the displayed text.

Prerequisites

Familiarity with the topics covered in this chapter.

Lab Setup

To run the solution to this lab, click this icon.

To see a demonstration of the solution to this lab, click this icon.

Estimated time to complete this lab: **45 minutes**.

Exercises

The following exercises provide practice working with the concepts and techniques covered in this chapter.

Exercise 1: Building the Framework for an SDI Application

In this exercise, you will build the simplest possible MFC application: an SDI frame window. All of the limited functionality in this application is provided by the AppWizard-generated classes.

Exercise 2: Adding a Time Pane to the Status Bar

In this exercise, you will add a new pane to the status bar and set its text to the current time. You will also add a command handler to provide updates to the status bar.

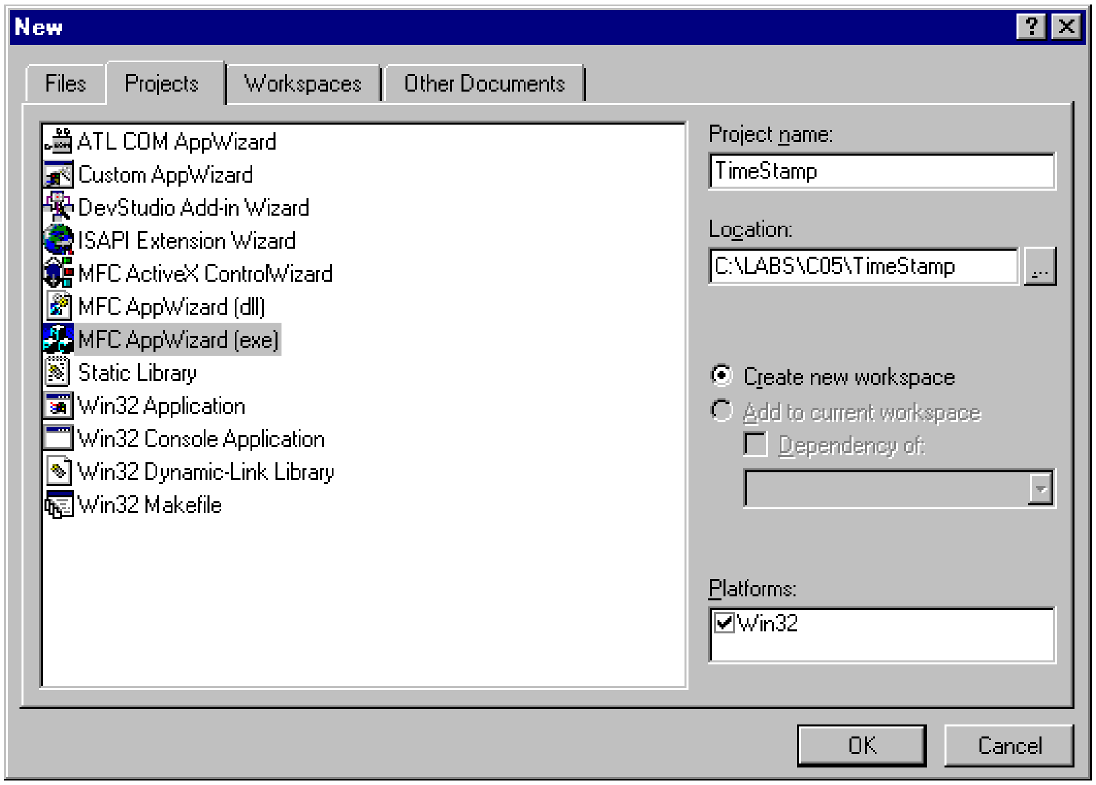
In this lab, you create a project. The first exercise creates a framework. The main exercise of modifying the status bar builds upon this framework. The completed code for these exercises is in \Labs\C05\Lab05\Xxx, where Xxx is the exercise number.

Exercise 1: Building the Framework for an SDI Application

In this exercise, you will build the simplest possible MFC application: an SDI frame window. All of the limited functionality in this application is provided by the AppWizard-generated classes.

➢ **Create a new AppWizard project**

1. Start Microsoft Developer Studio.
2. From the File menu, choose New.
3. In the Projects tab, choose MFC AppWizard (exe).
4. Name the project TimeStamp.



- ATL COM AppWizard
- Custom AppWizard
- DevStudio Add-in Wizard
- ISAPI Extension Wizard
- MFC ActiveX ControlWizard
- MFC AppWizard (dll)
- MFC AppWizard (exe)
- Static Library
- Win32 Application
- Win32 Console Application
- Win32 Dynamic-Link Library
- Win32 Makefile

Project name: TimeStamp

Location: C:\LABS\C05\TimeStamp

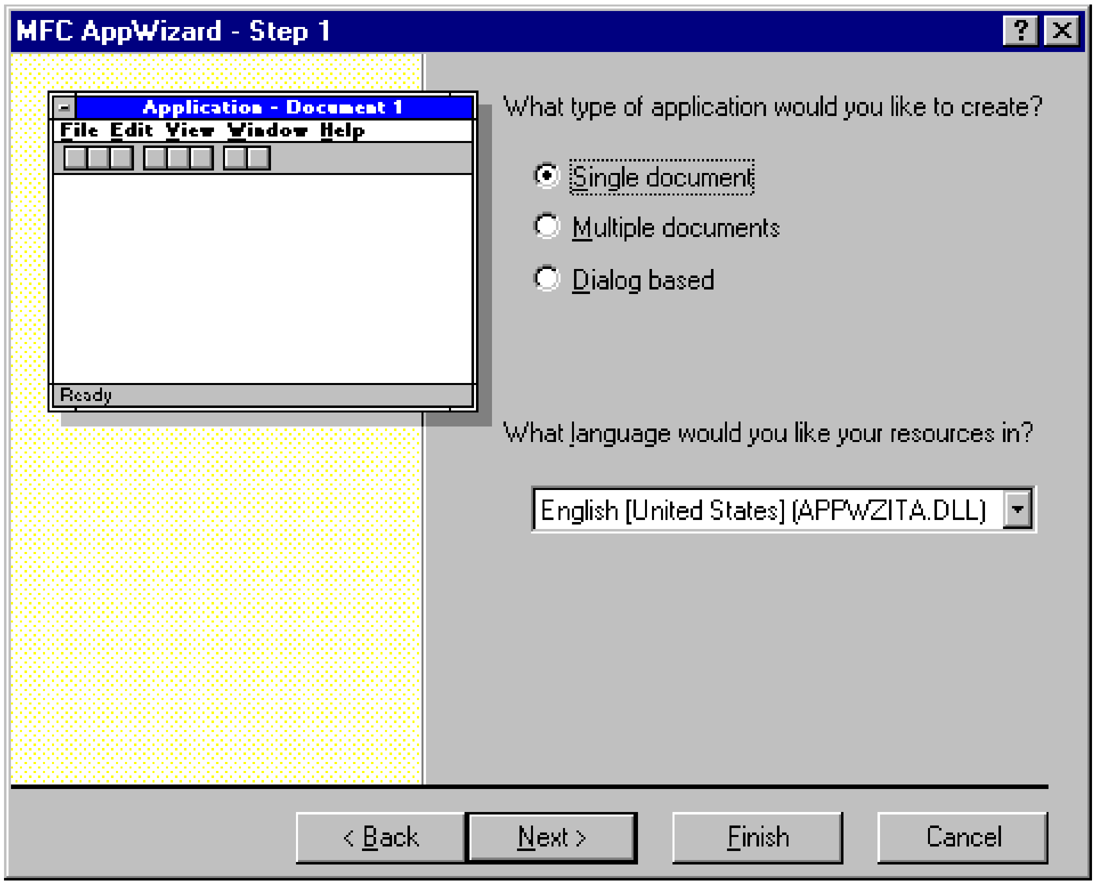
● Create new workspace

○ Add to current workspace

☐ Dependency of:

Platforms: ✓ Win32

5. Set the location for your project.
6. Click OK to create the new workspace.
7. MFC AppWizard will start. In Step 1, choose Single Document application and English language support. Click Next to go to the next page.



What type of application would you like to create?

[✓ Single document]

Multiple documents

Dialog based

What language would you like your resources in?

English [United States] (APPWZITA.DLL)

Note that Step 1 gives you three choices:

Single document	Single Document Interface (SDI): Only one document window is displayed at a time.
Multiple documents	Multiple Document Interface (MDI): Multiple document windows are displayed at a time.
Dialog-based	The application runs as a dialog box rather than a stand-alone window system.

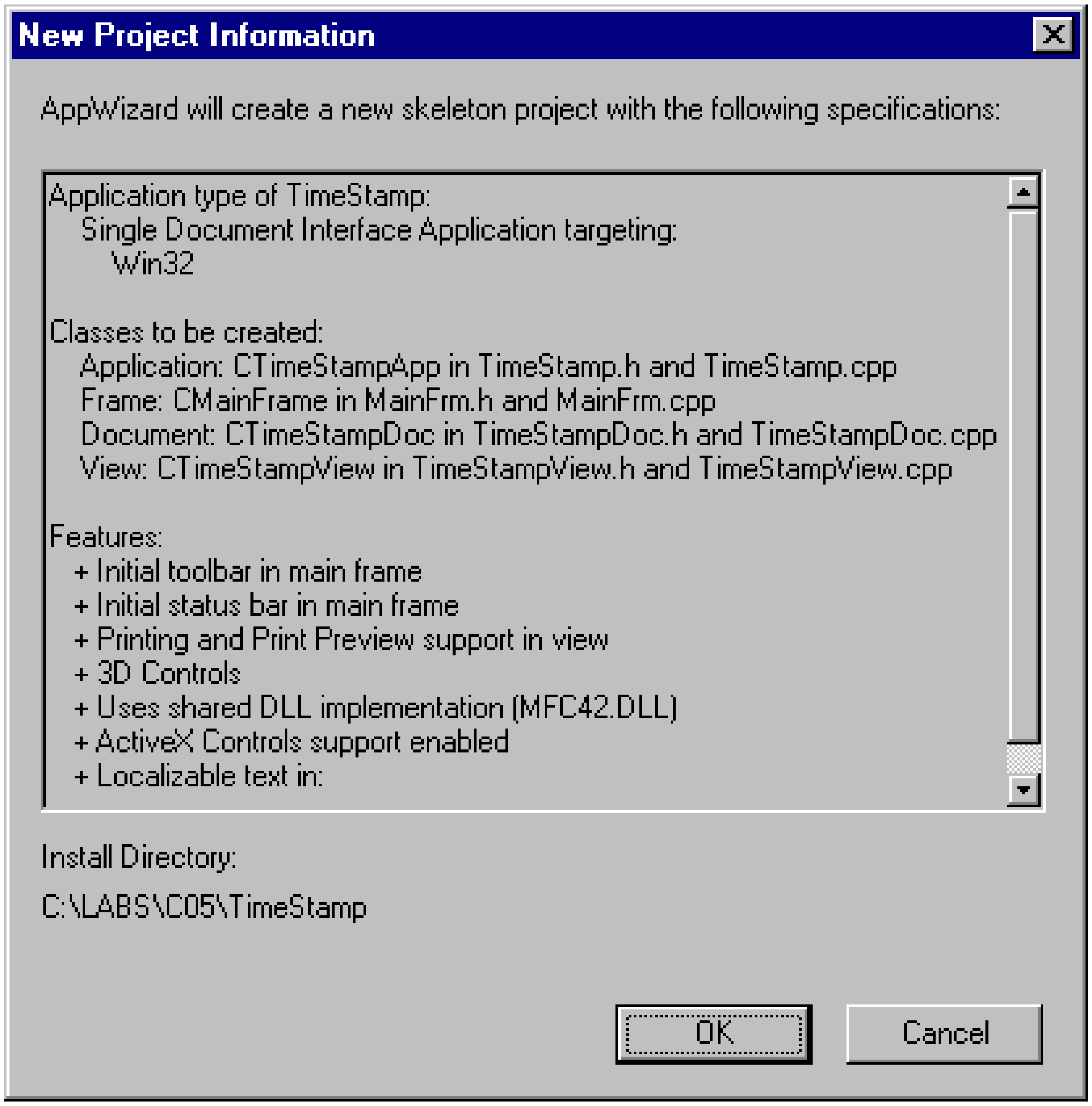
This lab focuses on the status bar, so there is no need for multiple documents. Choose the SDI .

8. Because no database support is needed, accept the default "None" for database support in Step 2.
9. In Step 3, accept the default (None). Do not check the ActiveX boxes since it is not supported in this application.
10. Accept the defaults in Step 4 (Docking toolbar, Initial status bar, Printing and print preview, 3D controls, and 4 files in the recent list). There are no changes in the Advanced options.
11. In Step 5, generate source file comments. Choose either static or shared DLL for MFC support.

Note When you choose to support MFC in a statically linked library, you can include the relevant code from the MFC libraries. Your .Exe is larger because of this. When you choose to support MFC as a shared DLL, your .Exe is smaller, but you need to include Mfc40.Dll as part of your distribution. For the purposes of this course, it makes sense to use shared DLLs to keep file size smaller.

---

12. In Step 6, accept the files and classes proposed in Step 6, and click Finish.
13. Visual C++ will display the New Project Information dialog box that summarizes your choices.



AppWizard will create a new skeleton project with the following specifications:

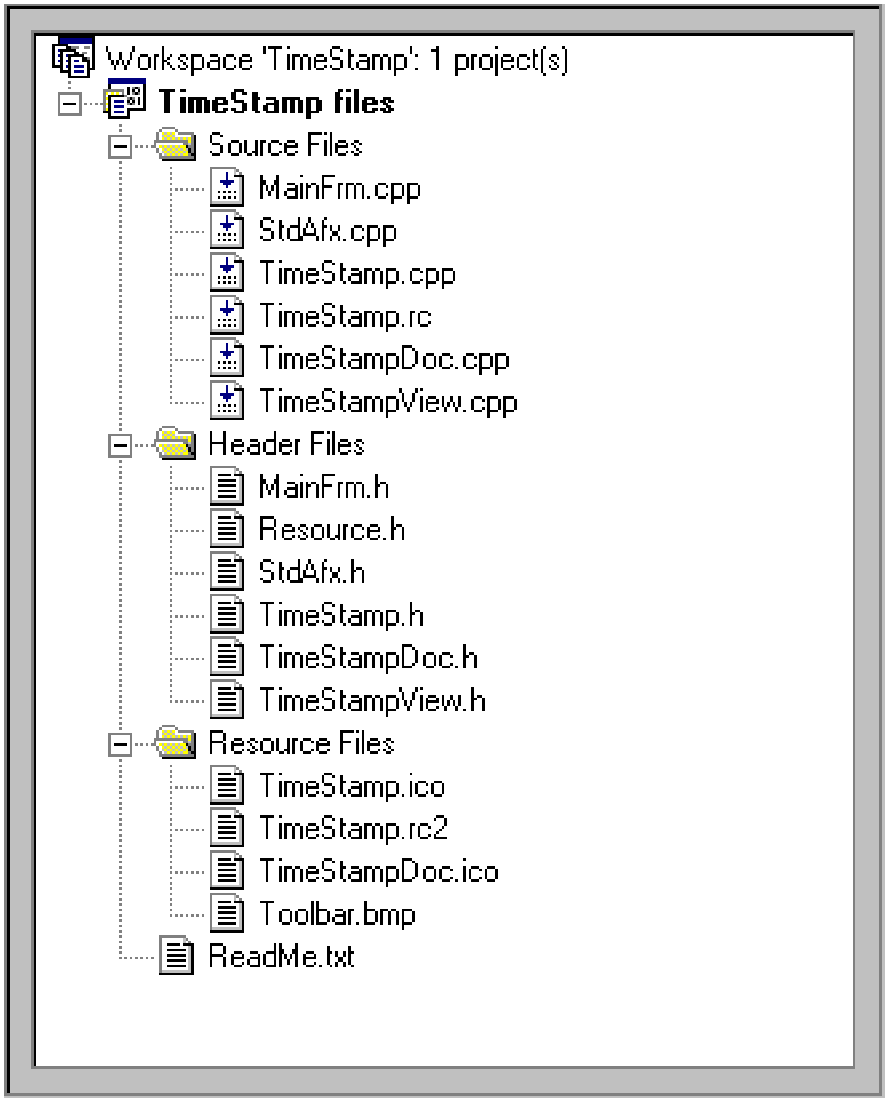
```
Application type of TimeStamp:
    Single Document Interface Application targeting:
        Win32

Classes to be created:
    Application: CTimeStampApp in TimeStamp.h and TimeStamp.cpp
    Frame: CMainFrame in MainFrm.h and MainFrm.cpp
    Document: CTimeStampDoc in TimeStampDoc.h and TimeStampDoc.cpp
    View: CTimeStampView in TimeStampView.h and TimeStampView.cpp

Features:
   + Initial toolbar in main frame
   + Initial status bar in main frame
   + Printing and Print Preview support in view
   + 3D Controls
   + Uses shared DLL implementation (MFC42.DLL)
   + ActiveX Controls support enabled
   + Localizable text in:
```

Install Directory:

C:\LABS\C05\TimeStamp

If you click Cancel, you return to the AppWizard to change your choices. If you click OK, AppWizard creates the application files for you.

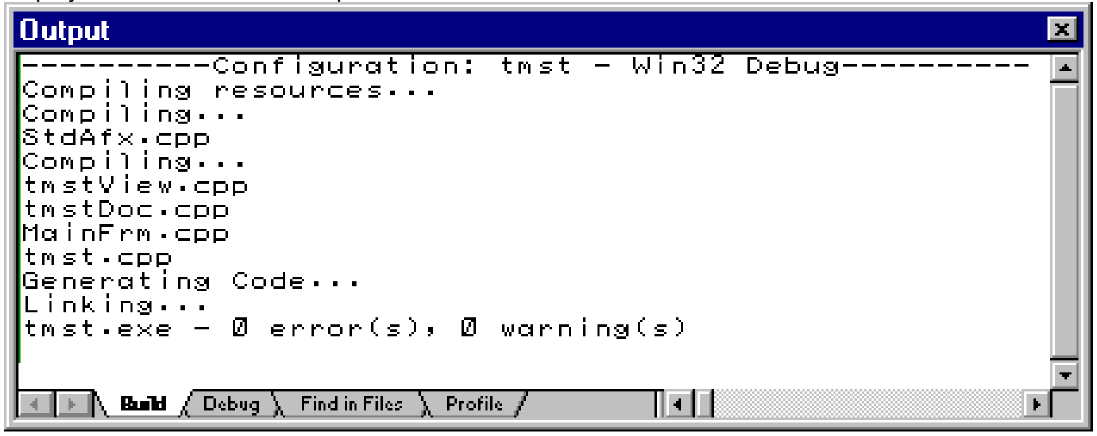
14. Click OK to return to the Developer Studio. Choose the file view icon to see the files that AppWizard created.



```
Workspace 'TimeStamp': 1 project(s)
TimeStamp files
    Source Files
        MainFrm.cpp
        StdAfx.cpp
        TimeStamp.cpp
        TimeStamp.rc
        TimeStampDoc.cpp
        TimeStampView.cpp
    Header Files
        MainFrm.h
        Resource.h
        StdAfx.h
        TimeStamp.h
        TimeStampDoc.h
        TimeStampView.h
    Resource Files
        TimeStamp.ico
        TimeStamp.rc2
        TimeStampDoc.ico
        Toolbar.bmp
    ReadMe.txt
```

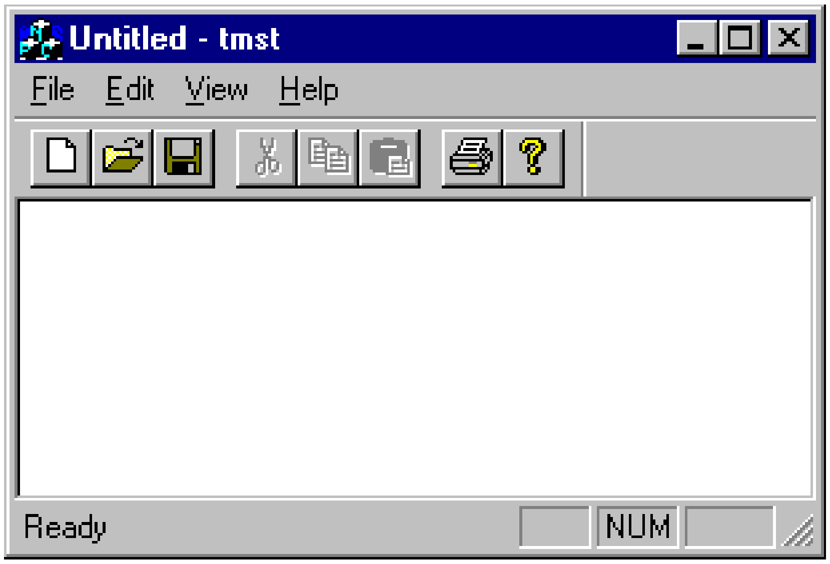
➤ **Build the project**

1. From the Build menu, choose Build TimeStamp.Exe, or press SHIFT+F8. The Developer Studio displays the status of the build process.



```
----------Configuration: tmst - Win32 Debug----------
Compiling resources...
Compiling...
StdAfx.cpp
Compiling...
tmstView.cpp
tmstDoc.cpp
MainFrm.cpp
tmst.cpp
Generating Code...
Linking...
tmst.exe - 0 error(s), 0 warning(s)
```

2. After the build is complete, from the Build menu, choose Execute TimeStamp.Exe or press CTRL+F5. TimeStamp will start.



There is not much functionality in the TimeStamp application right now, but many of the basics are in place: menus, toolbar, status bar, and window frame. These serve as the foundation for adding a new pane to the status bar, which you will do in Exercise 2.

The completed code for this exercise is in \Labs\C05\Lab05\Ex01.

Exercise 2: Adding a Time Pane to the Status Bar

Continue with the files you created in Exercise 1 or, if you do not have a starting point for this exercise, the code that forms the basis for this exercise is in \Labs\C05\Lab05\Ex01.

In this exercise, you add a new pane to the status bar and set its text to the current time. Use an UPDATE_COMMAND_UI handler to provide updates to the status bar.

CStatusBar includes support for adding panes to the status bar. The framework stores indicator information in an array with the leftmost indicator at position 0. When you create a status bar, you use an array of string IDs that the framework associates with the corresponding indicators. You can then use either a string ID or an index to access an indicator.

By default, the first indicator is "elastic”: It takes up the status bar length not used by the other indicator panes, so that the other panes are right-aligned.

If you examine **CMainFrame::On Create**, you can find the creation code for the status bar:

```
if (!m_wndStatusBar.Create(this) ||
    !m_wndStatusBar.SetIndicators(indicators,
        sizeof(indicators)/sizeof(UINT)))
```

CStatusBar::SetIndicators uses the strings identified by the constants in the indicators array to size and initialize the text that displays in the indicators. To add a pane, you need to add a string to the string table and a constant to the indicators array.

➤ Add a pane to the status bar

1. In the ResourceView, open the String Table folder and open the String Table resource.
2. Double-click the empty line at the end of the table, or select it and choose Properties from the Edit menu.
3. Set the ID of the string to ID_INDICATOR_TIME.
4. Set the Caption of the string to HH:MM AM. Note that if you use II:II II, your time display is clipped to a smaller space.
5. Save TimeStamp.Rc.

6. Open MainFrm.Cpp.
7. Add ID_INDICATOR_TIME to the indicators[ ] array. Save MainFrm.Cpp. The array follows:

```
static UINT indicators[] =
{
    ID_SEPARATOR,           // status line indicator
    ID_INDICATOR_CAPS,
    ID_INDICATOR_NUM,
    ID_INDICATOR_SCRL,
    ID_INDICATOR_TIME
};
```

➤ **Update the time pane**

Whenever the framework finishes processing all the pending messages for the MainFrame window, it sends itself a WM_IDLEUPDATECMDUI message. This message is one of the generators of the UPDATE_COMMAND_UI message that is sent by the system to CCmdTarget-derived objects such as **CStatusBar**. Use this message to force the update of the time pane.

1. Because the ClassWizard does not recognize the indicator ID that you created in the previous steps as a message target, you have to edit the message map by hand. Open MainFrm.Cpp and scroll to the **CMainFrame** message map.
2. After //}}AFX_MSG_MAP, but before the END_MESSAGE_MAP macro, add a map for UpdateCommandUI and ID_INDICATOR_TIME.

```
ON_UPDATE_COMMAND_UI(ID_INDICATOR_TIME, OnUpdateTime)
```

3. Save MainFrm.Cpp. The complete message map follows.

```
BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
    //{{AFX_MSG_MAP(CMainFrame)
        // NOTE - the ClassWizard will add and remove mapping macros here.
        //    DO NOT EDIT what you see in these blocks of generated code !
    ON_WM_CREATE()
    //}}AFX_MSG_MAP
    ON_UPDATE_COMMAND_UI(ID_INDICATOR_TIME, OnUpdateTime)
END_MESSAGE_MAP()
```

4. Open MainFrm.H. In the protected message map section, declare OnUpdateTime outside the //{{AFX_MSG block, and before the DECLARE_MESSAGE_MAP macro.

```
void OnUpdateTime(CCmdUI *pCmdUI)
```

5. Save MainFrm.H.
6. Open MainFrm.Cpp. Code the **OnUpdateTime** function by creating a CTime object and initializing it to the current system time as follows.

```
CTime   time = CTime::GetCurrentTime();
```

7. Use **CTime::Format** to format the time in HH:MM AM format and assign that string to a new **CString**.

```
CString sTime = time.Format ("%I:%M %p");
```

8. Set the text of the pane to the time.

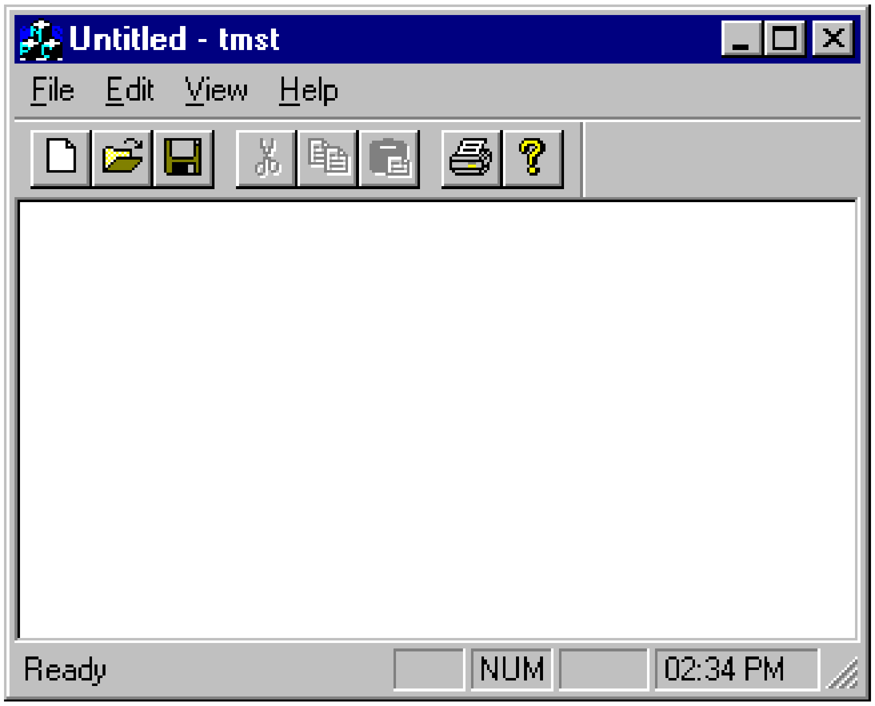
```
pCmdUI->SetText(sTime);
```

9. Save MainFrm.Cpp. The complete function is as follows.

```
void CMainFrame::OnUpdateTime(CCmdUI *pCmdUI)
{
    CTime   time = CTime::GetCurrentTime();
    CString sTime = time.Format ("%I:%M %p");
```

```
        // Now set the text of the pane.
        pCmdUI->SetText(sTime);
    }
```

10. Build and run the TimeStamp application. You will see the current time displayed in the fifth pane of the status bar.



➢ Use a timer to cause background updating

The TimeStamp application will show the time and update itself as long as there are messages for its frame window to process. However, TimeStamp is little more than a clock in a status bar. Without messages, an WM_IDLEUPDATECMDUI message is not sent to update the time pane. In this section, you can set up a timer for TimeStamp to process and update time.

1. Open MainFrm.Cpp. At the end of the **OnCreate** function and before the **return** statement, set a timer that will send a message once a second. The ID of this timer is irrelevant and can therefore be any value; there is no callback function.

```
        SetTimer(1234,1000, NULL);
```

2. Choose ClassWizard from the View menu, or press CTRL+W.
3. In the **CMainFrame** class, add a function for WM_TIMER. Even though you cannot add functionality to the default behavior, you must map the message through a function for the message pump to empty the queue and send the UPDATE_COMMAND_UI message. You can leave the **OnTimer** handler as provided or you can comment out the call to the default function.
4. Save MainFrm.Cpp. Build and run TimeStamp.Exe. You will notice that the timer pane updates properly whether or not TimeStamp.Exe is in the foreground or background and whether or not there has been activity in the window.

The completed code for this exercise is in \Labs\C05\Lab05\Ex02.