

WEEK

1

DAY 4

타이머 작업하기

규칙적으로 어떤 동작을 수행할 필요가 있는 애플리케이션을 만들어야 할 때가 있다. 단순히 1초 간격으로 상태바에 현재 시간을 표시하든가, 5분마다 복구 파일을 작성하던가 하는 용도 말이다. 사실 이러한 동작은 여러분이 날마다 사용하는 몇몇 애플리케이션에서는 일반적인 기능으로 돌아가는 것이다. 리소스 감시자(Resource Monitor)나 수행성능 감시자(Performance Monitor)처럼 특정한 리소스를 체크해야 하는 동작들도 규칙적으로 행해져야 하는 동작에 포함된다. 하지만, 방금까지 이야기한 것은 윈도우 운영체제에서 타이머를 사용하는 예의 빙산의 일각에 지나지 않는다.

이번 장에서는 타이머에 대해 공부해 보도록 하자.

- ❖ 비주얼 C++ 애플리케이션에서 타이머를 조절하고 사용하는 방법
- ❖ 다른 시간 간격을 가진 타이머를 여러 개 사용하는 방법
- ❖ 어떤 타이머인지 구별하는 방법
- ❖ 모든 비주얼 C++ 애플리케이션에 타이머를 부착시키는 방법

원도우 타이머에 대하여

원도우 타이머는 몇 밀리세컨드 간격으로 어떤 일을 하게끔 하는 신호를 발생시키는 매커니즘으로, 1개 이상의 타이머를 사용할 수 있다. 만일 어떤 타이머를 1,000밀리세컨드 간격으로 설정하면, 이 타이머는 1초 간격으로 발생된다. 타이머는 설정된 단위 시간 간격을 만날 때마다 WM_TIMER라는 메시지를 여러분의 애플리케이션으로 보내는데, 이 메시지를 처리할 수 있는 함수는 클래스위저드를 사용해서 추가할 수 있다.

- ✓ 타이머 이벤트가 애플리케이션의 이벤트 큐에 놓이는 경우는 이벤트 큐가 텅 비어 있고 애플리케이션이 휴지 기간(idle time : 애플리케이션이 아무 일도 하지 않는 때)에 있을 때 뿐이다. 원도우 운영체제는 애플리케이션의 이벤트 큐에 다른 메시지들이 득실거리고 있을 때는 절대로 타이머 이벤트 메시지를 넣지 않으며, 결과적으로 이때는 꾀같이 귀한 타이머 이벤트 몇 개를 잃어버리게 된다. 다시 정리하면, 원도우 운영체제는 이벤트 큐에 단 한 개의 타이머 이벤트 메시지를 넣으면, 애플리케이션이 바쁘게 돌아갈 때는 모든 타이머 이벤트 메시지를 보내지 않는다. 하지만, 얼마나 많은 메시지들을 잃었는가는 중요하지 않다. 여전히 메시지 큐에는 단 하나의 타이머 메시지만이 들어가니까 말이다.

타이머를 시동시키거나 중지시킬 때는 특정한 타이머의 ID(이 값은 정수값이다)를 설정해야 한다. 여러분의 애플리케이션은 이 ID를 사용해서 어떤 타이머가 발생되었는지를 결정하고, 해당 타이머를 시동시키거나 중지시킬 수 있는 것이다. 사실은 그만 하기로 하고 이제 실습에 들어가면서 머리 속에 잘 정리하도록 하자.

애플리케이션에 시계를 넣자

이번 장에서 만들 애플리케이션은 두 개의 타이머를 가지고 있다. 첫번째 타이머는 윈도우에 시계 기능을 넣기 위한 것으로, 애플리케이션이 실행되는 도중에는 계속 돌아간다. 두번째 타이머는 사용자가 시간 간격을ダイ얼로그 박스를 통해서 설정해 줄 수 있고, 사용자의 의지대로 시동시키거나 중지시킬 수 있다.

■ 프로젝트와 애플리케이션 골격을 만들자

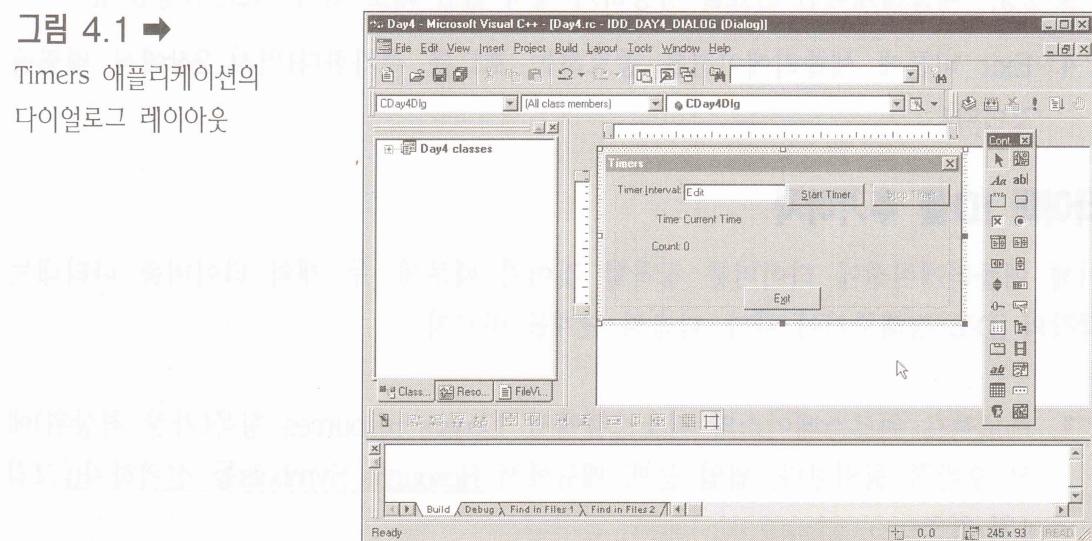
이번 장에서 만들 애플리케이션의 제작 과정은 3단계이다. 첫번째 단계에서는 전체 애플리케이션에 필요한 컨트롤을 놓고, 두번째 단계에서는 두 개의 타이머 중 시계 기능을 위한 처음 것을 추가할 것이며, 세번째 단계에서는 사용자가 원하는 대로 설정할 수 있는 타이머를 하나 더 추가할 것이다.

이제 시작하자.

- Timers란 이름의 새 프로젝트를 만든다. 지난 3개의 장에서 사용했던 똑같은 설정을 애플리케이션 위저드 설정으로 해주고, 애플리케이션의 타이틀만 Timers로 바꾸자.
- 다이얼로그 윈도우를 [그림 4.1]과 같이 꾸미자. 컨트롤 프로퍼티 정보는 [표 4.1]을 참고한다. 컨트롤의 프로퍼티를 설정할 때는 컨트롤에서 오른쪽 클릭한 다음, 문맥 메뉴에서 Properties를 선택한다는 사실을 아직 잊지는 않았으리라 믿는다.

그림 4.1 →

Timers 애플리케이션의
다이얼로그 레이아웃



[표 4.1] 컨트롤 프로퍼티 설정

컨트롤	프로퍼티	설정
정적 텍스트	ID	IDC_STATIC
	Caption	Timer & Interval:
에디트 박스	ID	IDC_INTERVAL
버튼	ID	IDC_STARTTIME
	Caption	&Start Timer
버튼	ID	IDC_STOPTIMER
	Caption	S&top Timer
	Disabled	체크
정적 텍스트	ID	IDC_STATIC
	Caption	Time:
정적 텍스트	ID	IDC_STATICTIME
	Caption	<u>Current Time</u>
정적 텍스트	ID	IDC_STATIC
	Caption	Count:
정적 텍스트	ID	IDC_STATICCOUNT
	Caption	0
버튼	ID	IDC_EXIT
	Caption	E&xit

3. 2장, “애플리케이션 컨트롤 사용하기”에서 했던 대로 탭 순서를 설정한다.
4. Exit 버튼에 애플리케이션을 종료하는 코드를 추가한다(역시 2장에서 배웠을 것이다).

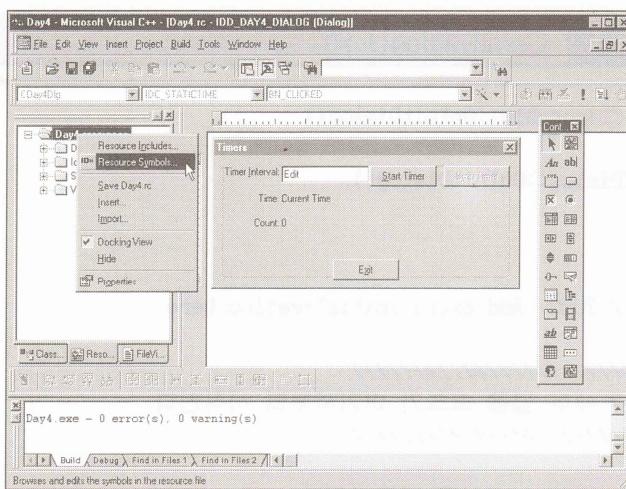
■ 타이머 ID를 추가하자

이제 애플리케이션에 타이머를 장착할 것이기 때문에, 두 개의 타이머를 나타내는 각각의 ID도 정해주어야 한다. 다음의 단계를 따르자.

1. 프로젝트 워크스페이스의 리소스 뷰에서 Timers resources 항목(가장 최상위)에서 오른쪽 클릭한다. 팝업 문맥 메뉴에서 Resource Symbols를 선택하자([그림 4.2] 참조).

그림 4.2 ➔

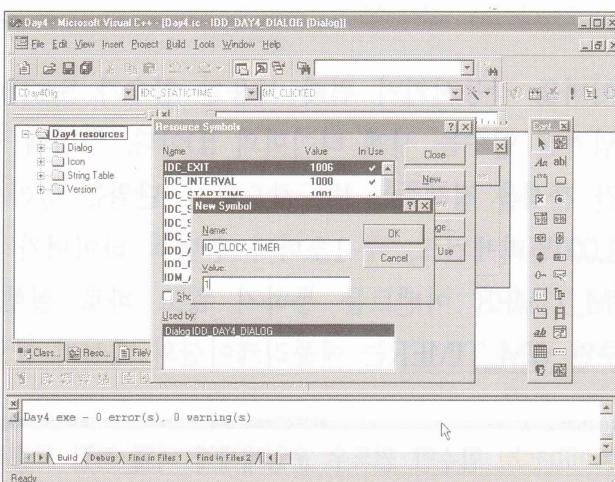
리소스에 관한
팝업 문맥 메뉴



2. Resource Symbols 다이얼로그에서 New 버튼을 클릭한다.
3. [그림 4.3]과 같이 New Symbol 다이얼로그에서 심볼 이름(symbol name)으로 ID_CLOCK_TIMER를, 값(value)으로 1을 입력하자.

그림 4.3 ➔

리소스 심볼을
새로 추가한다.



4. 2번 단계와 3번 단계를 반복해서, 심볼로 ID_COUNT_TIMER, 값으로 2를 추가 입력한다.
5. Close 버튼을 클릭해서 Resource Symbols 다이얼로그를 닫는다. 이제 두 개의 타이머 ID를 여러분의 애플리케이션에서 사용할 수 있게 될 것이다.

■ 시계 타이머를 작동시키자

시계 타이머를 작동시키기 위해, 이전의 두 장에서 보아왔던 OnInitDialog() 함수를 찾아서 [리스트 4.1]과 같이 코드를 써넣도록 하자.

리스트 4.1 OnInitDialog() 함수

```

1: BOOL CTimersDlg::OnInitDialog()
2: {
3:     CDialog::OnInitDialog();
4: .
5: .
6: .
7:     // TODO: Add extra initialization here
8: .
9:     /////////////////
10:    // 새로 넣을 코드가 여기서부터 시작된다
11:    /////////////////
12: .
13:    // 시계 타이머를 작동시킨다.
14:    SetTimer(ID_CLOCK_TIMER, 1000, NULL);
15: .
16:     /////////////////
17:    // 새로 넣을 코드는 여기서 끝난다
18:    /////////////////
19: .
20:    return TRUE; // return TRUE unless you set the focus to a control
21: }

```

여기서 타이머를 작동시키기 위해 사용한 함수가 SetTimer() 함수이다. 이 함수에 첫번째로 넘겨진 인수는 시계 타이머의 ID이다. 두번째 인수는 타이머가 신호를 발생시킬 시간 간격을 정해주는 빈도값으로서, 단위는 밀리세컨드이다. 지금 작성한 예제에서는 1,000밀리세컨드, 즉 1초마다 한번씩 타이머가 발생되도록 하였다. 세번째 인수는 WM_TIMER 이벤트를 통하지 않고 바로 실행될 콜백 함수의 주소인데, NULL을 주면 WM_TIMER는 애플리케이션의 메시지 큐로 보내라는 뜻이다.

Note

콜백(Callback) 함수란 윈도우 운영체제에 의해 직접 호출되는 함수를 일컬으며, 여러분이 작성해 둔다. 어떤 서브시스템이 이 콜백 함수를 호출하며, 왜 호출했는가에 따라 독특한 인수 정의를 가지며, 함수 정의를 미리 해둔 후에는 원하는 어떤 것이라도 할 수 있게 된다.

어떤 콜백 함수를 인수로 받아들이는 윈도우 함수에 그 함수의 주소를 넘기는 것으로 콜백 함수의 작동 준비는 끝이다. 이후 윈도우 운영체제가 이 콜백 함수를 호출해야 할 상황을 만나기만 하면 설정된 콜백 함수는 바로바로 호출된다.

■ 시계 타이머 이벤트를 처리하자

이제 타이머는 작동되고 말았다. 일은 저질렀으니 뒷처리도 해주어야 하지 않겠는가? 지금 할 일은 타이머 메시지를 처리하는 코드를 추가하는 것이다.

1. 클래스위저드를 사용해서 IDC_STATICTIME 컨트롤에 CString 타입의 m_sTime이란 이름을 가진 변수를 추가한다.
2. 클래스위저드를 사용해서 WM_TIMER 메시지에 대한 함수를 CTimersDlg 개체에 추가한다.
3. OnTimer() 함수를 [리스트 4.2]와 같이 만들자.

리스트 4.2 OnTimer() 함수

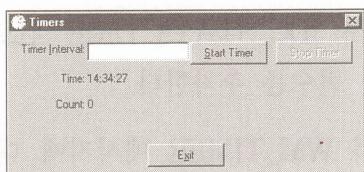
```

1: void CTimersDlg::OnTimer(UINT nIDEvent)
2: {
3:     // TODO: Add your message handler code here and/or call default
4:
5:     ///////////////////////
6:     // 새로 넣을 코드가 여기서부터 시작된다
7:     /////////////////////
8:
9:     // 현재 시간을 얻어낸다.
10:    CTime curTime = CTime::GetCurrentTime();
11:
12:    // 현재 시간을 표시한다
13:    m_sTime.Format('%d:%d:%d",
14:        curTime.GetHour(),
15:        curTime.GetMinute(),
16:        curTime.GetSecond());
17:
18:    // 다이얼로그를 갱신한다
19:    UpdateData(FALSE);
20:
21:    /////////////////////
22:    // 새로 넣을 코드는 여기서 끝난다
23:    /////////////////////
24:    CDialog::OnTimer(nIDEvent);
25: }
```

CTime 클래스의 인스턴스를 선언함과 동시에 현재의 시스템 시간값으로 초기화했다. 다음 m_sTime 변수를 Format() 함수(CString 클래스의 멤버 함수)를 사용해서 현재 시간을 시:분:초 형식으로 만들었고, 마지막으로 다이얼로그 윈도우에 새로 고쳐진 시간값을 출력하였다. 지금 이 애플리케이션을 컴파일하고 실행시키면 [그림 4.4]와 같이 다이얼로그 윈도우의 한 가운데에 1초 간격으로 현재 시간이 표시되는 모습을 확인할 수 있다.

그림 4.4 →

애플리케이션 디자인로그에서
작동되는 시계



애플리케이션에 타이머를 하나 더 추가하자

주지하다시피, 애플리케이션에 타이머를 추가하는 일은 황당할 정도로 쉬운 일이다. SetTimer()를 어디선가 호출한 다음, OnTimer() 함수에 원하는 코드를 써주면 되는 그만이기 때문이다. 하지만, 타이머를 하나 더 덧붙인다고 할 때에는 약간의 부가작업이 필요하다.

컨트롤에 변수가 필요하다

여러분의 애플리케이션에 두번째 타이머를 덧붙이기 전에, 우선 몇 개의 컨트롤에 변수를 더 물려 두어야 한다. 시계용 타이머를 사용할 때에는 시간값을 표시하기 위한 문자열 변수가 하나 필요했었다. 이번에는 어떤 컨트롤에 어떤 변수를 추가해야 할까? [표 4.2]를 보도록 하자.

[표 4.2] 컨트롤에 물려줄 변수들

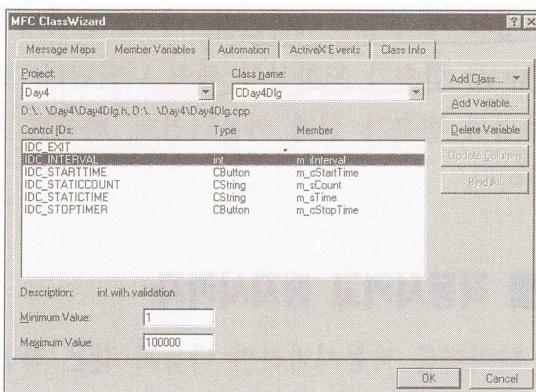
컨트롤	이름	카테고리	타입
IDC_STATICCOUNT	m_sCount	Value	CString
IDC_INTERVAL	m_iInterval	Value	int
IDC_STARTTIME	m_cStartTime	Control	CButton
IDC_STOPTIMER	m_cStopTime	Control	CButton

앞의 변수는 클래스위저드를 사용해서 모두 추가하자. 이제는 무엇을 하는고 하니, 다음과 같다.

- 클래스위저드의 Member Variable 탭에서 m_iInterval 변수를 선택하고 나서, [그림 4.5]와 같이 아래쪽에 나타난 두 개의 에디트 박스, 즉 Minimum Value와 Maximum Value에 각각 1과 100000을 넣도록 하자.

그림 4.5 ▶

변수의 범위를 설정한다.



2. 프로젝트 워크스페이스의 클래스뷰에서, 3장에서 배운 방법을 써서 CTimersDlg 클래스에 멤버 변수를 하나 추가하는데, 변수 타입은 int로 하고, 변수 이름은 m_iCount로 할 것이며, 액세스 지정자로는 Private를 선택하자.
3. 클래스위저드를 다시 띄운 다음, IDC_INTERVAL 컨트롤 ID(에디트 박스이다)에 대해 EN_CHANGE 이벤트의 함수를 추가한다. 성공했으면 [리스트 4.3]에 나온대로 이 함수를 작성하자.

리스트 4.3 OnChangeInterval() 함수

```

1: void CTimersDlg::OnChangeInterval()
2: {
3:     // TODO: If this is a RICHEDIT control, the control will not
4:     // send this notification unless you override the CDlg::OnInitialUpdate()
5:     // function and call CRichEditCtrl().SetEventMask()
6:     // with the EN_CHANGE flag ORed into the mask.
7:
8:     // TODO: Add your control notification handler code here
9:
10:    /////////////////
11:    // 새로 넣을 코드가 여기서부터 시작된다
12:    ///////////////
13:
14:    // 컨트롤에 물려 둔 변수를 갱신한다
15:    UpdateData(TRUE);
16:
17:    /////////////////
18:    // 새로 넣을 코드는 여기서 끝난다
19:    ///////////////
20: }
```

방금 전에 클래스위저드로 했던 타이머 간격을 조정하는 변수의 범위를 설정하는 작업이 선행되었을 경우, 사용자가 해당 에디트 박스에 그 범위에 해당되지 않는 값을

넣으면 자동으로 경고 메시지가 나타난다. 이 경고 메시지는 OnChangeInterval() 함수 안의 UpdateData() 함수가 도화선이 되어 나타난다. 프로젝트 워크스페이스를 통해 마지막으로 추가된 변수가 실제 카운터인데, 타이머 이벤트가 발생할 때마다 이 값이 증가되는 것이다.

■ 카운트 타이머를 작동시키고 중지시키자

여러분의 두번째 타이머를 작동시키려면 다음과 같은 작업을 해야 한다.

- ❖ m_iInterval 변수의 초기화
- ❖ IDC_STARTTIME 버튼이 클릭되었을 때 타이머 작동시키기
- ❖ m_iCount 변수를 증가시키고 타이머 이벤트가 발생할 때마다 다이얼로그를 갱신하기
- ❖ IDC_STOPTIMER 버튼이 클릭되었을 때 타이머 중지시키기

이제 추가 기능의 구현을 시작해 보도록 하자.

1. OnInitDialog() 함수를 [리스트 4.4]와 같이 만들자.

리스트 4.4 바꿔어진 OnInitDialog() 함수

```

1: BOOL CTimersDlg::OnInitDialog()
2: {
3:     CDialog::OnInitDialog();
4:
5:
6:
7:     // TODO: Add extra initialization here
8:
9:     ///////////////////////
10:    // 새로 넣을 코드가 여기서부터 시작된다
11:    /////////////////////
12:
13:    // 카운터 간격을 초기화 한다
14:    m_iInterval = 100;
15:
16:    // 다이얼로그를 갱신한다
17:    UpdateData(FALSE);
18:
```

```

19: // 시계 타이머를 작동한다.
20: SetTimer(ID_CLOCK_TIMER, 1000, NULL);
21:
22: /////////////////
23: // 새로 넣을 코드는 여기서 끝난다
24: /////////////////
25:
26: return TRUE; // return TRUE unless you set the focus to a control
27: }

```

2. 클래스위저드를 사용해서 IDC_STARTTIME 버튼에 대한 BN_CLICKED 메시지의 함수를 추가한다. [리스트 4.5]와 같이 OnStarttime() 함수의 몸체를 작성하자.

리스트 4.5 OnStarttime() 함수

```

1: void CTimersDlg::OnStarttime()
2: {
3:     // TODO: Add your control notification handler code here
4:
5:     /////////////////
6:     // 새로 넣을 코드가 여기서부터 시작된다
7:     /////////////////
8:
9:     // 컨트롤에 물려둔 변수를 갱신한다
10:    UpdateData(TRUE);
11:
12:    // 카운트를 초기화한다.
13:    m_iCount = 0;
14:    // 카운트 값을 가지고 문자열을 서식화한다.
15:    m_sCount.Format("%d", m_iCount);
16:
17:    // 디아일로그를 갱신한다
18:    UpdateData(FALSE);
19:    // 타이머를 작동시킨다
20:    SetTimer(ID_COUNT_TIMER, m_iInterval, NULL);
21:
22:    /////////////////
23:    // 새로 넣을 코드는 여기서 끝난다
24:    /////////////////
25: }

```

3. 클래스위저드를 사용해서 IDC_STOPTIMER 버튼에 대한 BN_CLICKED 메시지의 함수를 추가하고, OnStoptimer() 함수를 [리스트 4.6]과 같이 만들자.

리스트 4.6 OnStoptimer() 함수

```

1: void CTimersDlg::OnStoptimer()
2: {
3:     // TODO: Add your control notification handler code here
4:
5:     /////////////////
6:     // 새로 넣을 코드가 여기서부터 시작된다
7:     /////////////////
8:
9:     // 타이머를 중지시킨다
10:    KillTimer(ID_COUNT_TIMER);
11:
12:    /////////////////
13:    // 새로 넣을 코드는 여기서 끝난다
14:    /////////////////
15: }
```

4. 이젠 OnTimer() 함수를 고쳐서 [리스트 4.7]과 같이 만들자.

리스트 4.7 OnTimer() 함수

```

1: void CTimersDlg::OnTimer(UINT nIDEvent)
2: {
3:     // TODO: Add your message handler code here and/or call default
4:
5:     /////////////////
6:     // 새로 넣을 코드가 여기서부터 시작된다
7:     /////////////////
8:
9:     // 현재 시간을 얻는다
10:    CTime curTime = CTime::GetCurrentTime();
11:
12:    // 어느 타이머가 이 이벤트를 발생시켰는가?
13:    switch (nIDEvent)
14:    {
15:        // 네놈 시계냐?
16:        case ID_CLOCK_TIMER:
17:            // 그렇다면 현재 시간을 표시하시지.
18:            m_sTime.Format("%d:%d:%d", curTime.GetHour(),
19:                           curTime.GetMinute(),
20:                           curTime.GetSecond());
21:            break;
22:        // 카운트 타이머이냐?
23:        case ID_COUNT_TIMER:
24:            // 카운트값을 증가하시지
25:            m_iCount++;
26:            m_iCount++;
27:            // 카운트값을 서식화하고 표시하도록.
```

```

27:         m_sCount.Format("%d", m_iCount);
28:         break;
29:     }
30:
31:     // 디아일로그를 갱신한다
32:     UpdateData(FALSE);
33:
34:     ///////////////////////
35:     // 새로 넣을 코드는 여기서 끝난다
36:     /////////////////////
37:
38:     CDialog::OnTimer(nIDEvent);
39: }

```

OnInitDialog() 함수에서 이미 m_iInterval 변수의 초기화 코드를 추가하여 100부터 시작하게 만들어 두었다. 이 초기화 값은 UpdateData() 함수에 의해 디아일로그 윈도우에 바로 반영된다.

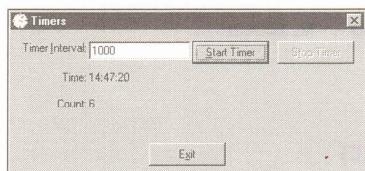
OnStarttime() 함수에서는 일단 컨트롤과 컨트롤에 물려둔 변수의 내용을 일치시켜서 현재의 m_iInterval 변수의 값을 얻어낼 수 있게 하였다. 다음 m_iCount 변수를 0으로 설정하고, 이 값을 CString 타입의 변수인 m_sCount를 사용해서 서식화하였다. m_sCount의 값은 이후에 디아일로그 윈도우로 다시 보내어진다. 마지막으로 여러분이 해준 일은 ID_COUNT_TIMER란 ID와 m_iInterval 변수에서 얻어낸 간격 값을 가지고 타이머를 작동시킨 것이다.

OnStopTimer() 함수에서는 그냥 이 타이머를 중지시켜 주기만 하면 끝이다. 이때 사용하는 함수가 KillTimer() 함수이고, 중지시킬 타이머의 ID를 인수로 받는다.

점점 재미있어 지려고 하는 부분이 OnTimer() 함수이다. 여기서는 여전히 시계 타이머 이벤트를 처리하는 코드가 남아 있는데, 사실 여러 개의 타이머를 애플리케이션에 붙였다고 해도 타이머 이벤트를 처리하는 함수는 이것 하나이기 때문에 전혀 신경쓸 필요는 없다. 여기에 카운터 타이머를 처리하는 코드를 덧붙이기 위해서는 단지 어떤 타이머 때문에 OnTimer()로 프로그램의 제어권이 왔는지만 구별해 주면 그만이다. 이때 사용하는 문장이 switch와 case이며, 구별 기준은 타이머의 ID이다. 카운터 타이머는 바로 여기에 끼어들어갔는데, 카운터가 증가되었고 m_sCount 변수가 새 값으로 바뀌어졌다. 이 상태에서 애플리케이션을 컴파일하고 실행시키면 [그림 4.6]과 같이 타이머의 시간 간격을 직접 설정하고 타이머를 작동시킬 수 있을 것이다.

그림 4.6 →

애플리케이션 디자인로그에서
작동되는 카운터



■ Stop 버튼에도 힘을 몰아주자

여러분의 애플리케이션을 실행시키면, 하찮은 문제 딱 하나를 제외하고는 모두 잘 돌아감을 알 수 있다. 즉, 두번째 타이머를 작동시키고 나서 중지시킬 방법이 없는 것이다. 이번 장의 내용을 충실히 잘 따라왔다면, 아마도 Stop Timer 버튼이 비활성화되었을 것이다. 타이머를 중지시키기 전에 활성화시켜 두어야 하는데 말이다.

한 가지 더 덧붙인다면 타이머를 중지시키는 버튼을 활성화시키는 동시에 타이머를 작동시키는 버튼은 비활성화시켜야 한다. 그리고 타이머가 중지되었다면 이 상황을 반대로 하는 것도 잊어서는 안된다. 컨트롤을 활성화시키고 비활성화시키는 것은 2장에서 배운 대로 하면 되며, 여기에 조금만 수정을 해주도록 하자.

기억해 둘 것은, 컨트롤에 변수를 물려두는 과정에서 타이머를 작동하고 중지하는 버튼에도 변수를 물려두었어야 한다는 것이다. 또한 보통의 변수가 아닌 컨트롤 (Control) 타입의 변수이어야 한다. GetDlgItem() 함수에 컨트롤의 ID를 넘겨서 포인터를 얻어내는 방법도 있지만, 컨트롤 변수로 직접 제어하는 방법을 사용하기로 하자. [리스트 4.8]은 바뀐 OnStarttime() 함수와 OnStoptimer() 함수이다.

리스트 4.8 OnStarttime() 함수와 OnStoptimer() 함수를 고친 결과

```

1: void CTimersDlg::OnStarttime()
2: {
3:     // TODO: Add your control notification handler code here
4:
5:     ///////////////////////////////
6:     // 새로 넣을 코드가 여기서부터 시작된다
7:     ///////////////////////////////
8:
9:     // 변수를 갱신한다
10:    UpdateData(TRUE);
11:
12:    // 카운트를 초기화한다
13:    m_iCount = 0;
14:    // 카운트 값을 서식화한다
15:    m_sCount.Format("%d", m_iCount);

```

```

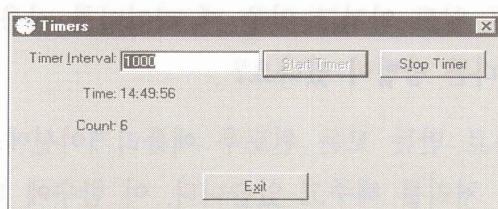
16: // 파일로그를 갱신한다
17: UpdateData(FALSE);
18: // 타이머를 작동시킨다
19: SetTimer(ID_COUNT_TIMER, m_iInterval, NULL);
20:
21:
22: // 버튼을 활성화시킨다
23: m_cStopTime.EnableWindow(TRUE);
24: // 버튼을 비활성화시킨다
25: m_cStartTime.EnableWindow(FALSE);
26:
27: /////////////////
28: // 새로 넣을 코드는 여기서 끝난다
29: ///////////////
30: }
31:
32: void CTimersDlg::OnStoptimer()
33: {
34: // TODO: Add your control notification handler code here
35:
36: ///////////////
37: // 새로 넣을 코드가 여기서부터 시작된다
38: ///////////////
39:
40: // 타이머를 중지한다
41: KillTimer(ID_COUNT_TIMER);
42:
43: // 버튼을 비활성화시킨다
44: m_cStopTime.EnableWindow(FALSE);
45: // 버튼을 활성화시킨다
46: m_cStartTime.EnableWindow(TRUE);
47:
48: ///////////////
49: // 새로 넣을 코드는 여기서 끝난다
50: ///////////////
51: }

```

이제 완성된 것 같다. 애플리케이션을 컴파일하고 실행시키면, [그림 4.7]과 같이 카운터 타이머를 작동하고 중지할 수 있을 것이다. 타이머 간격을 조절하고 그 차이를 눈으로 확인해 보자. 우리의 시계는 변함없이 파일로그 한가운데에서 돌아가고 있다.

그림 4.7

완성된 애플리케이션



요약

이번 장에서 여러분은 윈도우 운영체제에 내장된 타이머를 애플리케이션에 추가하여 다른 용도로 사용하고 제어하는 방법에 대해 공부해 보았는데, 한 개 이상의 타이머를 동시에 붙여서 다른 동작을 하게도 조작해 보았다.

이어질 장들에서는 다이얼로그 박스를 통해 사용자의 입력을 받아 애플리케이션의 동작을 조정하는 방법과 애플리케이션에 메뉴를 추가하는 방법을 공부해 나갈 것이다. 두 가지가 끝난 다음에는 텍스트와 폰트에 대해 공부하기로 하자.

Q&A

- ① 제 애플리케이션에 설정할 수 있는 타이머 간격의 한계가 어느 정도인지 알고 싶어요.
- ② 설정할 수 있는 타이머 간격의 범위는 약 55밀리세컨드에서 2^{32} -1밀리세컨드(약 49일하고 반나절)입니다.
- ③ 애플리케이션에 동시에 붙일 수 있는 타이머는 몇 개나 되나요?
- ④ 경우에 따라 다릅니다. 윈도우 운영체제에서 돌아가는 모든 애플리케이션에 사용할 수 있는 타이머의 수는 제한되어 있어요. 타이머를 몇 개 사용하지 않기 때문에 사용할 수 있는 타이머의 수가 충분하다고 해도, 하나의 애플리케이션에서 막나가는 동작을 수행하면 운영체제도 이상해집니다. 운영체제의 권한에 따라 타이머의 사용을 할 수 없게 되는 애플리케이션은 여러분의 애플리케이션일 수도 있고 다른 애플리케이션일 수도 있지요. 일반적인 규칙은, 동시에 두 개나 세 개 이상의 타이머가 사용되었다면 여러분의 애플리케이션 설계를 다시 고려해서 좀더 적은 수의 타이머를 사용하도록 만들라는 것입니다.
- ⑤ 애플리케이션이 아무 일도 안하고 있을 때 타이머를 사용하지 않고 어떤 작업을 수행하게 할 수 있는 다른 방법이 있나요?
- ⑥ 그럼요. 비주얼 C++로 만든 모든 윈도우 애플리케이션에는 OnIdle()이란 함수가 있어서 이에 해당하는 처리를 해주고 있습니다. 이 함수에 대해서는 18장, “멀티태스킹 - 동시에 여러 개의 작업을 수행하기”에서 학습하게 될 것입니다.

실습해 보기

“실습해 보기” 절에서는 배운 것을 확인하는 퀴즈와 이를 활용해서 응용력을 높이기 위한 연습문제를 풀어볼 기회를 가질 수 있게 될 것이다. 퀴즈와 연습문제의 답은 부록 B, “퀴즈 및 연습문제 해답”에 있으며, 정직한 사람만이 성공한다는 말을 더 불리고 싶다.

■ 퀴즈

1. 리소스 심볼에 두 개의 타이머 ID를 추가함으로써 무엇을 한 것인가?
2. 애플리케이션에 ID 두 개를 추가하는 다른 방법은 무엇인가?
3. OnTimer() 함수에서 두 개의 타이머를 구분하려면 어떻게 할까?
4. 타이머가 1초 간격으로 설정되고 여러분의 애플리케이션이 타이머 이벤트 메시지를 받지 못하게 1분 동안 계속 다른 일을 해왔다면, 얼마나 많은 타이머 이벤트가 발생되었을까?

■ 연습문제

여러분의 애플리케이션을 개선시켜서 카운터 타이머가 시작되었을 때 시계 타이머가 리셋되어 카운터 타이머와 동일한 간격으로 동작하게 해보자. 카운터 타이머가 중지되었을 때 시계 타이머는 다시 1초 간격으로 되돌아오도록 해야 한다.