

WEEK

1

DAY 7

텍스트와 폰트로 작업하기

단도 직입적으로 말해서, 대부분의 윈도우 애플리케이션에서의 폰트 설정에 관한 걱정은 기우에 불과하다. 단순히 사용될 폰트를 설정해 주지 않아도, 윈도우 운영체제는 디폴트 폰트를 알아서 제공한다. 만일 특별한 폰트가 하나 필요하다면 Dialog Properties 대이얼로그 박스를 사용해서 특정한 대이얼로그 윈도우에 사용될 폰트를 선택하면 된다.

하지만, 간혹(또는 자주) 여러분의 애플리케이션에 사용되는 폰트를 조정해 주어야 할 필요가 있게 마련이다. 사용되고 있는 폰트를 바꾸거나 사용자로 하여금 특정한 환경에서 사용자로 하여금 폰트를 고르도록 해야 될 때도 있다. 이러한 상황을 위해 이번 장의 공부가 필요 한 것이다. 여러분이 배울 내용은 다음과 같다.

- ◆ 사용 가능한 폰트 리스트를 구축하는 방법
- ◆ 사용할 폰트를 설정하는 방법
- ◆ 실행중에 폰트를 바꾸는 방법

폰트를 찾아 사용해 보자

폰트 작업에 있어서 알아두어야 할 첫번째 사실 중 하나는, 모든 시스템에 똑같은 폰트가 설치되어 있지는 않다는 사실이다. 폰트는 필요에 따라 간편하게 윈도우 운영체제에 설치하고 제거할 수 있으며, 원하는 폰트가 무엇이든지 간에 사용자의 취향에 맞게 폰트를 구성할 수 있다. 만일 시스템에 존재하고 있지 않은 폰트를 쓰려고 요구하면 윈도우 운영체제는 자신에 설치된 폰트 중에 요구된 폰트와 가장 흡사한 폰트나 시스템 디폴트 폰트를 골라준다.

따라서, 여러분이 할 수 있는 일은 운영체제에게 “어떤 폰트를 사용할 수 있소?”라고 물어보는 것이다. 운영체제가 이에 관한 답을 해주면, 이 정보를 바탕으로 어떤 폰트를 사용할 것인지를 결정할 수 있거나 사용자에게 주어진 범위 내의 폰트를 선택하게 할 수 있다. 사용 가능한 폰트가 무엇인지를 운영체제에게 물을 때, 나열될 폰트의 타입을 제한할 수 있으며, 여러 가지 속성을 기반으로 한 여러 폰트 혹은 모든 폰트를 리스트로 만들어 내도록 할 수도 있다.

사용 가능한 폰트를 나열하자

현재 시스템에 설치되어 있는 사용 가능한 폰트의 리스트를 얻어내려면 Win32 API 함수를 사용해야 하는데, EnumFontFamiliesEx()라고 불리는 함수가 이 목적에 사용된다. 이 함수는 윈도우 운영체제에게 “당신에 설치된 폰트 리스트를 원하오”라고 알려주는 역할을 한다. EnumFontFamiliesEx()를 당장 사용해서 사용 가능한 폰트의 리스트를 주루룩 얻어내고 싶은 마음이 굴뚝같겠지만, 일단은 어떻게 이 리스트를 줄 수 있는지를 이해하고 넘어가야 한다.

콜백 함수에 대하여

EnumFontFamiliesEx() 함수의 중요한 인수 중 하나는 윈도우 운영체제에 의해 호출되는 다른 함수의 주소로서, 이 함수는 콜백(callback) 함수라고 일컫는다. 윈도우 운

영체체에서 돌아가는 대부분의 나열용 함수에는 콜백 함수의 주소를 넘겨주는 것이 보통인데, 나열된 리스트의 한 항목을 찾을 때마다 한번씩 콜백 함수가 호출되기 때문이다. 말하자면, 시스템에 설치된 각각의 폰트를 받아 리스트로 구축해 내는 함수를 여러분의 애플리케이션 안에 스스로 만들어 주어야 한다는 뜻이다.

콜백 함수를 만들어서 각 폰트 정보를 받아 리스트를 구축하는 일을 스스로 해야 한다고 해서 아무 방법으로나 콜백 함수를 정의할 수 없다. 모든 콜백 함수는 윈도우 API에 이미 정의되어 있기 때문에, 용도에 맞춰진 타입의 콜백 함수를 사용해야 한다. 즉, 어떻게 함수가 정의되어 있고, 인수 리스트는 어떠하며, 반환해야 하는 값의 타입은 어떤지가 이미 내정되어 있다는 뜻이다. 단지 함수의 이름과 내부 동작만은 설정되어 있지 않기 때문에 골격만이 준비된 셈이다. 아무튼 폰트 리스트를 얻어내기 위해 사용되는 함수 타입은 `EnumFontFamProc`이다.

EnumFontFamiliesEx 함수

`EnumFontFamiliesEx()` 함수는 시스템에서 사용 가능한 폰트 리스트를 요구할 때 호출하며, 다섯 개의 인수를 받는다. 이 함수의 전형적인 사용 예를 보기로 하자.

```
// 디바이스 컨텍스트를 잡아낸다
CCClientDC dc (this);
// LOGFONT 구조체를 선언한다
LOGFONT lLogFont;
// 문자 셋을 설정한다
lLogFont.lfCharSet = DEFAULT_CHARSET;
// 모든 폰트의 리스트를 받아내기로 결정한다
lLogFont.lfFaceName[0] = NULL;
// 히브리어나 아랍어가 아니면 0이어야 한다
lLogFont.lfPitchAndFamily = 0;
// 폰트 부류(Font Family)을 나열하도록 한다
::EnumFontFamiliesEx((HDC) dc, &lLogFont,
(FONTENUMPROC) EnumFontFamProc, (LPARAM) this, 0);
```

이 함수의 첫번째 인수는 디바이스 컨텍스트(Device Context)로서, 여기서는 `CCClientDC` 클래스의 인스턴스가 되었다. 윈도우 운영체제에서 실행되는 모든 애플리케이션에는 디바이스 컨텍스트가 붙어있다고 해도 과언이 아니다. 디바이스 컨텍스트는 애플리케이션에서 사용될 수 있고, 사용될 수 없는 그래픽에 관련된 것에 관한 운영체제의 정보가 많이 들어 있다.

두번째 인수는 LOGFONT 구조체의 포인터이다. 이 구조체에는 여러분이 리스트 구축에 필요하다고 여긴 폰트에 관한 정보가 들어 있으며, 어떤 문자 집합을 사용할 것인가 또는 특정 폰트 부류 내의 모든 폰트를 리스트에 넣을 것인가를 이 구조체 안에서 설정할 수 있다. 만일 시스템 상의 모든 폰트를 리스트에 넣고 싶으면 이 인수에 NULL을 주면 끝이다.

세번째 인수는 폰트 리스트를 구축하기 위해 사용될 콜백 함수의 주소이다. 주소라고 해서 거창한 것은 아니고, 단순히 콜백 함수의 이름을 주면 된다. 비주얼 C++ 컴파일러가 컴파일시에 함수의 이름을 함수 주소로 바꾸어 놓기 때문에 걱정하지 않아도 된다. 하지만, 인수로 넘길 때 콜백 함수의 타입으로 캐스팅하는 일은 빼먹어서는 안된다.

네번째 인수는 콜백 함수로 넘겨질 LPARAM 값이다. 이 인수는 윈도우 운영체제에 의해 사용되는 것이 아니라 콜백 함수가 폰트 리스트를 구축하는데 필요한 문맥 정보이다. 앞에 쓰인 예제에서 사용된 LPARAM 값은 코드가 실행중인 윈도우의 포인터이며, 콜백 함수는 이 포인터를 사용해서 폰트 리스트의 구축에 필요한 구조체를 액세스할 수 있는 것이다. 뭐 다른 자료구조를 LPARAM으로 줄 수도 있다. 폰트의 링크드 리스트의 첫번째 노드 혹은 다른 구조체로의 포인터라든지 말이다.

다섯번째와 마지막 인수는 항상 0이다. 이 예약값은 윈도우 운영체제의 차기 버전에서 사용될 것 같아서 만들어둔 것이다. 아무튼 현재는 0만을 써야 하며, 다른 값을 넣어주었을 때 발생되는 오동작에 대해서는 책임지지 않겠다.

EnumFontFamProc 함수 타입

여러분의 콜백 함수를 스스로 만들 때는 C++ 클래스의 멤버가 아닌 전역 함수 형태로 정의해야 한다. 전형적인 EnumFontFamProc 함수의 형식은 다음과 같다.

```
int CALLBACK EnumFontFamProc(
    LPENUMLOGFONT lpelf,
    LPNEWTEXTMETRIC lpntm,
    DWORD nFontType,
    long lParam)
{
    // 다이얼로그 윈도우의 포인터로 캐스팅한다
    CMyDlg* pWnd = (CMyDlg*) lParam;

    // 리스트 박스에 폰트 이름을 넣는다
```

```

pWnd->m_ctlFontList.AddString(lpelf->elfLogFont.lfFaceName);
// 나열 동작을 계속하기 위해 1을 반환한다
return 1;
}

```

이 함수의 첫번째 인수는 ENUMLOGFONTEX 구조체의 포인터이다. 이 구조체는 폰트의 이름, 스타일, 스크립트를 포함한 폰트의 논리적 속성에 대한 정보를 모두 가지고 있다. 아무튼 여러분은 똑같은 이름이지만 다른 스타일을 가진 폰트를 리스트에 담을 수 있으며, 보통 글씨체, 볼드체, 이탤릭체 폰트를 담을 수도 있다.

두번째 인수는 NEWTEXTMETRICEX 구조체의 포인터로서, 폰트의 높이, 폭, 폰트 주변의 공간 등의 물리적인 속성에 관한 정보를 담고 있다. 폰트 자체의 크기를 확대 축소하는 것이 가능하기 때문에 이 구조체에 나온 값을 모두 상대적인 값임을 명심하자.

세번째 인수는 폰트의 태입을 설정하는 플래그로서, 아래에 나온 값을 조합해서 가질 수 있다

- ❖ DEVICE_FONTPROP
- ❖ RASTER_FONTPROP
- ❖ TRUETYPE_FONTPROP

마지막으로, 네번째 인수는 EnumFontFamiliesEx() 함수에 넘겨지는 long 타입의 값이다. 여기서는 폰트 리스트가 구축되고 있는 디바이스 윈도우의 포인터를 사용했다. 이 값을 디바이스 윈도우의 포인터로 캐스팅하고 난 후에는 바로 리스트 박스 컨트롤을 액세스해서 폰트 이름을 추가할 수 있게 된다.

이 함수에서 반환된 값은 폰트 리스트의 구축을 계속할 것인가를 나타내는데, 0이 반환되면 윈도우 운영체제는 더 이상 콜백 함수를 호출하지 않으며, 1이 반환되면 콜백 함수가 계속 호출되어 폰트 리스트의 구축이 계속 진행된다.

■ 폰트를 어떻게 만들고 사용하나

애플리케이션에서 어떤 폰트를 하나 사용하려면 일단 CFont 클래스의 인스턴스를 메모리에 생성하자. 다음 CreateFont() 멤버 함수를 호출하여 사용할 폰트의 여러 가

지 정보, 즉 폰트의 크기, 스타일, 방향(orientation)을 설정하여 폰트를 생성한다. 일단 폰트가 생성되었으면, 그 폰트를 사용하게 될 컨트롤이나 윈도우 개체의 SetFont() 함수를 사용해서 폰트를 바꿀 수 있다. 간략한 예를 다음에 보였다.

```
CFont m_fFont; // 사용될 폰트 변수를 선언한다

// 사용될 폰트를 생성한다
m_fFont.CreateFont(12, 0, 0, 0, FW_NORMAL,
    0, 0, DEFAULT_CHARSET, OUT_CHARACTER_PRECIS,
    CLIP_CHARACTER_PRECIS, DEFAULT_QUALITY, DEFAULT_PITCH +
    FF_DONTCARE, m_sFontName);

// 디스플레이 영역에 대해 폰트를 설정한다
m_ctlDisplayText.SetFont(&m_fFont);
```

Tip

앞의 코드에서 쓰인 CFont 변수는 이 코드가 놓인 클래스의 멤버 변수로 선언되었지만, 어떻게 선언되는가를 보이기 위해 한번 위에다가 써보았다. 다짐해두지만, 이 변수는 함수 안에서 지역 변수로 선언되거나 사용되어서는 안된다.

딱 두 개의 함수만 사용되니까 언뜻 보기에는 무척 단순해 보이지만, 첫번째로 쓰인 CreateFont() 함수는 황당할 정도로 많은 인수가 필요해 보인다. 인수가 많다는 것은 그만큼 CreateFont() 함수가 엄청난 유연성을 가지고 있다는 뜻이다. 아무튼 일단 폰트를 생성하고 나면, CWnd 클래스의 멤버인 SetFont() 함수에 이 폰트를 인수로 넘김으로써 모든 작업이 끝난다. SetFont()는 CWnd 클래스의 멤버이기 때문에 이 클래스에서 상속받은 모든 윈도우 클래스나 컨트롤 클래스에서 사용할 수 있다. 즉, 비주얼 C++ 애플리케이션 내에서 눈에 보이는 구성요소라면 이 방법을 써도 된다는 뜻이다.

어쨌거나, CreateFont()는 매우 중요한 함수이기 때문에 각각의 인수를 살펴보는 것도 결코 무익하지 않을 것이다. 다음을 보자.

```
BOOL CreateFont(
    int nHeight,
    int nWidth,
    int nEscapement,
    int nOrientation,
    int nWeight,
    BYTE bItalic,
    BYTE bUnderline,
    BYTE cStrikeOut,
    BYTE nCharSet,
```

```

    BYTE nOutPrecision,
    BYTE nClipPrecision,
    BYTE nQuality,
    BYTE nPitchAndFamily,
    LPCTSTR lpszFaceName);

```

첫번째 인수인 nHeight는 폰트의 높이이다. 이 논리값은 나중에 물리적인 값으로 바뀐다. 만일 이 값이 0인 경우에는 운영체제가 알아서 적합한 값을 골라주며, 0보다 크거나 작은 값일 경우에는 절대값이 디바이스 단위로 변환된다(기준의 차이는 있다). -10과 10이 기본적으로 같은 높이라는 사실을 이해해두기 바란다.

두번째 인수인 nWidth는 폰트내 문자의 평균 폭을 가리킨다. 이 논리값은 높이값과 마찬가지로 나중에 물리적인 값으로 바뀐다.

세번째 인수인 nEscapement는 텍스트(줄)가 표시될 경사를 나타내는데, 반시계 방향으로 0.1도의 단위로 설정된다. 만약 텍스트를 아래에서 위 방향으로 수직 출력을 하고 싶다면 이 값에 900을 넣어주면 되며, 보통처럼 왼쪽에서 오른쪽으로 수평 출력을 하고 싶다면 0을 주자.

네번째 인수인 nOrientation은 폰트 내의 각각의 문자가 표시되는 방향을 나타내며, 측정 방식은 세번째 인수와 동일하지만, 텍스트 줄 전체의 각도가 아니라 각 문자의 각도임을 명심하자. 이를테면 문자들을 모두 뒤집어서 표시하고 싶으면 1800을, 옆으로 눕혀서 표시하고 싶으면 900을 주도록 하자.

다섯번째 인수인 nWeight는 폰트의 가중치, 즉 두께이다. 0부터 1000까지의 값을 가지며, 간편하게 쓸 수 있도록 상수로 정의되어 있다. [표 7.1]을 보도록 하자.

[표 7.1] 폰트 가중치 상수

상수	값
FW_DONTCARE	0
FW_THIN	100
FW_EXTRALIGHT	200
FW_ULTRALIGHT	200
FW_LIGHT	300

상수	값
FW_NORMAL	400
FW_REGULAR	400
FW_MEDIUM	500
FW_SEMIBOLD	600
FW_DEMIBOLD	600
FW_BOLD	700
FW_EXTRABOLD	800
FW_ULTRABOLD	800
FW_BLACK	900
FW_HEAVY	900

실제적인 가중치 변환은 폰트에 전적으로 달려있다. 어떤 폰트는 FW_NORMAL, FW_REGULAR, FW_BOLD 가중치만을 가지고 있는 경우도 있으니까 말이다. 만일 FW_DONTCARE을 넣어주면 디폴트 가중치가 사용된다.

여섯번째 인수인 bItalic은 폰트를 이탤릭으로 쓰느냐의 여부를 결정한다. 불리언 타입의 성질을 가진다. 즉, TRUE는 이탤릭으로 쓰겠다는 뜻이며, FALSE는 이탤릭으로 쓰지 않겠다는 뜻이다.

일곱번째 인수인 bUnderline은 폰트에 밑줄을 그을 것인지 말 것인지를 결정한다. 역시 불리언 타입의 성질을 가지기 때문에 여섯번째 인수와 비슷한 의미이다.

여덟번째 인수인 cStrikeOut는 폰트의 가운데를 가로지르는 선을 그을 것인지를 결정한다. 역시 불리언 타입의 성질을 가진다.

아홉번째 인수인 n CharSet은 폰트의 문자 집합을 설정한다. [표 7.2]에 나온 상수 중 하나를 사용하도록 한다.

여러분의 애플리케이션이 돌아가는 시스템이 다른 문자 집합을 사용할 수 있으며, OEM 문자 집합은 시스템마다 다르기 때문에, 혹시 깨름칙한 문자 집합을 사용하고 있다면 출력용 문자열을 조작하는 일이 자칫 위험을 초래할 수도 있으니까 문자열을 그대로 사용해 주기 바란다.

[표 7.2] 폰트 문자 집합을 설정하는 상수

상수	값
ANSI_CHARSET	0
DEFAULT_CHARSET	1
SYMBOL_CHARSET	2
SHIFTJIS_CHARSET	128
OEM_CHARSET	255

열번째 인수인 nOutPrecision은 요구된 폰트의 높이, 폭, 문자 방향, 텍스트 경사, 피치에 얼마나 근접하게 일치시키느냐를 설정하는 값인데, 역시 상수값으로 내정되어 있다.

◆ OUT_CHARACTER_PRECIS

◆ OUT_DEFAULT_PRECIS

◆ OUT_DEVICE_PRECIS

◆ OUT_RASTER_PRECIS

◆ OUT_STRING_PRECIS

◆ OUT_STROKE_PRECIS

◆ OUT_TT_PRECIS

OUT_DEVICE_PRECIS, OUT_RASTER_PRECIS, OUT_TT_PRECIS 값은 동일한 이름의 여러 개의 폰트가 있을 때 어떤 폰트를 선택할 것인가를 결정한다. 즉, OUT_TT_PRECIS를 사용했다면 같은 이름의 트루타입 폰트와 래스터 폰트가 있을 때 트루타입의 폰트를 선택하게 된다. 사실 OUT_TT_PRECIS 값은 요구된 폰트가 트루타입 버전을 지원하지 않을 때조차도 트루타입 폰트를 사용하도록 요청한다.

열한번째 인수인 nClipPrecision은 디스플레이 영역의 바깥으로 벗어난 문자를 어떻게 클리핑할 것인가를 설정한다. 역시 상수로 정의되어 있으며, 다음의 값을 가질 수 있다.

❖ CLIP_CHARACTER_PRECIS

❖ CLIP_DEFAULT_PRECIS

❖ CLIP_ENCAPSULATE

❖ CLIP_LH_ANGLES

❖ CLIP_MASK

❖ CLIP_STROKE_PRECIS

❖ CLIP_TT_ALWAYS

이들 값은 OR 연산자로 조합될 수 있다.

열두번째 인수인 nQuality는 출력 품질과 GDI(Graphics Device Interface : 그래픽 디바이스 인터페이스)가 논리 폰트 속성이 물리 폰트에 얼마나 근접하도록 할 것인가를 설정한다. 이 인수에 사용할 수 있는 값은 다음과 같다.

❖ DEFAULT_QUALITY

❖ DRAFT_QUALITY

❖ PROOF_QUALITY

열세번째 인수인 nPitchAndFamily는 폰트의 피치와 부류를 설정하며, 피치값과 부류 값을 OR한 형태로 넘겨주는 것이 보통이다. 사용 가능한 피치는 다음과 같다.

❖ DEFAULT_PITCH

❖ VARIABLE_PITCH

❖ FIXED_PITCH

다음에 나오는 값은 사용 가능한 부류이다.

❖ FF_DECORATIVE

❖ FF_DONT CARE

◆ FF_MODERN

◆ FF_ROMAN

◆ FF_SCRIPT

◆ FF_SWISS

폰트 부류(Font Family)은 폰트의 외양을 일반적으로 설명하는 이름이다. 요구된 폰트가 시스템에 존재하고 있지 않을 때는 대체 폰트를 선택하기 위해 폰트 부류를 사용할 수 있다.

마지막 인수인 lpszFacename은 사용될 폰트의 이름을 담고 있는 표준 C 스타일의 문자열이다. 이 이름은 EnumFontFamProc 콜백 함수에서 얻어낸 정보를 사용하면 뒷북치지 않고 정확하게 사용할 수 있을 것이다

폰트를 사용하자

이제 애플리케이션을 만드는 시간이 돌아왔다. 이번 장에서 만들어 볼 것은 표시되는 폰트 리스트 중에서 사용자가 선택할 수 있도록 하는 애플리케이션으로, 입력된 텍스트를 선택된 폰트를 사용해서 표시할 수 있다.

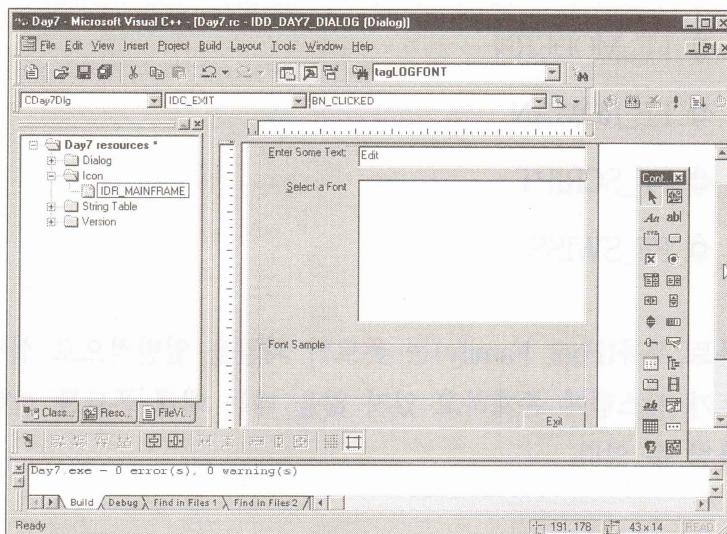
■ 애플리케이션 골격을 만들자

언제나 시작은 비슷한 것에서 출발하기 마련이다. 다음의 단계를 따르자.

1. MFC AppWizard(exe) 타입의 프로젝트를 새로 만들고, 이름은 Day7이라고 짓는다.
2. 애플리케이션 위저드의 각 단계에 지금까지 만들었던 애플리케이션에서 해준 설정을 그대로 해주고, 애플리케이션의 타이틀만 Fonts로 바꾼다.
3. [표 7.3]을 참고해서 [그림 7.1]과 같이 메인 디아얼로그를 꾸미자.

그림 7.1 →

메인 다이얼로그 레이아웃



[표 7.3] 컨트롤 프로퍼티 설정

컨트롤	프로퍼티	설정
정적 텍스트	ID	IDC_STATIC
	Caption	&Enter Some Text:
에디트 박스	ID	<u>IDC_ESAMPTEXT</u>
정적 텍스트	ID	IDC_STATIC
	Caption	&Select a Font
리스트 박스	ID	<u>IDC_LFONTS</u>
그룹 박스	ID	IDC_STATIC
	Caption	Font Sample
정적 텍스트	ID	<u>IDC_DISPLAYTEXT</u>
(그룹 박스 안에다가 두 고, 그룹 박스의 크기에 맞 게 늘이자)	Caption	<u>Empty string</u>
푸시 버튼	ID	IDC_EXIT
	Caption	E&xit

4. 클래스위저드를 사용해서, [표 7.4]를 참고하면서 다이얼로그 상의 컨트롤에 변수를 물려두자.

[표 7.4] 컨트롤에 물려줄 변수

컨트롤	이름	카테고리	타입
IDC_DISPLAYTEXT	m_ctlDisplayText	Control	CStatic
	m_strDisplayText	Value	CString
IDC_LFONTS	m_ctlFontList	Control	CListBox
	m_strFontName	Value	CString
IDC_ESAMPTTEXT	m_strSampText	Value	CString

5. IDC_EXIT 버튼에 함수를 연결하고 애플리케이션을 종료하는 코드를 넣어두자.
어떻게 하는지는 알고 있으리라 믿는다.

■ 폰트 리스트를 구축하자

여러분의 시스템에 설치된, 사용 가능한 폰트의 리스트를 구축하기 위해서는 일단 각각의 리스트를 얻어내어 다이얼로그 윈도우 상의 리스트 박스에 추가하는 동작을 수행해 줄 콜백 함수를 만들어야 한다. Day7Dlg.h 헤더 파일을 열고 앞부분쯤에 [리스트 7.1]에 나온 함수 선언을 추가하자. 이 함수는 절대 비주얼 C++에서 지원되는 도구를 사용해서 넣어줄 수 없다. 직접 손으로 입력해야 한다.

리스트 7.1 Day7Dlg.h 헤더 파일에 추가한 콜백 함수 선언문

```

1: #if _MSC_VER > 1000
2: #pragma once
3: #endif // _MSC_VER > 1000
4:
5: int CALLBACK EnumFontFamProc(LPENUMLOGFONT lpelf,
6: LPNEWTEXTMETRIC lpntm, DWORD nFontType, long lParam);
7:
8: /////////////////////////////////
9: // CDay7Dlg dialog
10:
11: class CDay7Dlg : public CDialog
12: {
13: .
14: .

```

헤더 파일에 콜백 함수의 선언문을 추가한 다음에는 Day7Dlg.cpp 소스 코드 파일을 열고 파일의 끝부분에 커서를 놓고 [리스트 7.2]에 나온 함수 정의를 차근히 입력하자.

리스트 7.2 소스 파일에 추가한 콜백 함수의 몸체

```

1: int CALLBACK EnumFontFamProc(LPENUMLOGFONT lpelf,
2: LPNEWTEXTMETRIC lpntm, DWORD nFontType, long lParam)
3: {
4:     // 다이얼로그 윈도우의 포인터로 캐스팅한다
5:     CDay7Dlg* pWnd = (CDay7Dlg*) lParam;
6:
7:     // 폰트 이름을 리스트 박스에 추가한다
8:     pWnd->m_ctlFontList.AddString(lpelf->elfLogFont.lfFaceName);
9:     // 동작을 계속하기 위해 1을 반환한다
10:    return 1;
11: }

```

콜백 함수의 준비는 끝났다. 이젠 윈도우 운영체제에게 폰트 리스트를 요구하는 함수를 추가할 순서이다. 다음의 단계를 따르자.

1. 프로젝트 워크스페이스의 클래스뷰 탭을 선택한다.
2. CDay7Dlg 클래스를 선택하고 오른쪽 클릭한 다음, 팝업 메뉴에서 Add Member Function을 선택한다.
3. 함수 타입은 void, 함수 선언은 FillFontList로 해주고, 액세스 지정자는 Private로 정한다. OK 버튼을 클릭해서 다이얼로그를 닫으면 함수 추가가 끝난다.
4. [리스트 7.3]에 나온 대로 이 함수를 만들자.

리스트 7.3 FillFontList 함수

```

1: void CDay7Dlg::FillFontList()
2: {
3:     LOGFONT lf;
4:
5:     // LOGFONT 구조체를 초기화한다
6:     lf.lfCharSet = DEFAULT_CHARSET;
7:     strcpy(lf.lfFaceName, "");
8:     // 리스트 박스를 썩 청소한다
9:     m_ctlFontList.ResetContent();
10:    // 디바이스 컨텍스트를 잡아 낸다
11:    CClientDC dc (this);
12:    // 폰트 부류를 나열한다.
13:    ::EnumFontFamiliesEx((HDC) dc, &lf,
14: (FONTENUMPROC) EnumFontFamProc, (LPARAM) this, 0);
15: }

```

5. OnInitDialog() 함수로 가서 [리스트 7.4]와 같이 FillFontList 함수를 호출하는 문장을 넣어준다.

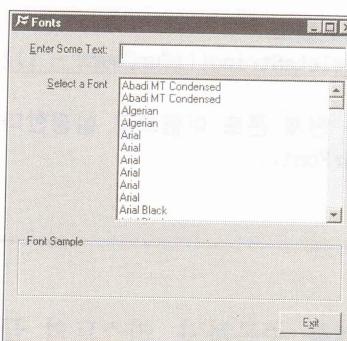
리스트 7.4 OnInitDialog() 함수

```
1: BOOL CDay7Dlg::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:    // 리스트 박스를 폰트로 채운다
14:    FillFontList();
15:    .
16:    ///////////////
17:    // 새로 넣을 코드는 여기서 끝난다
18:    ///////////////
19:    .
20:    return TRUE; // return TRUE unless you set the focus to a control
21: }
```

이제 여러분의 애플리케이션을 컴파일하고 실행시키면, 현재 시스템에 설치된 사용 가능한 폰트의 이름이 모두 리스트 박스에 채워짐을 알 수 있을 것이다. 하지만, 여러분이 원하지 않은 한 가지도 짜증스럽게 끼어들어가 있다. [그림 7.2]를 보면 리스트 박스에 똑같은 엔트리가 상당히 많이 들어가 있는데, 중복되는 것을 없애고 폰트마다 한 줄씩만 표시되게 하면 좋을 것 같다.

그림 7.2 →

시스템에 설치된 모든 폰트가 나열된다.



EnumFontFamiliesEx() 함수 호출은 원래 동기적(synchronous)임을 여기서 밝혀 두겠다. 무슨 뜻인가 하니, 이 함수는 시스템 내의 모든 폰트가 콜백 함수 안에서 리스트 박스에 추가되고 나서야 종료된다는 뜻이다. 따라서 FillFontList() 함수에 몇 가지 처리를 해주어서, 리스트 박스가 채워지고 난 직후에 중복된 모든 엔트리를 제거하도록 한다. [리스트 7.5]와 같이 FillFontList() 함수를 만들도록 하자.

리스트 7.5 수정된 FillFontList 함수

```

1: void CDay7Dlg::FillFontList()
2: {
3:     int iCount;           // 폰트의 개수
4:     int iCurCount;        // 현재 폰트
5:     CString strCurFont;  // 현재 폰트 이름
6:     CString strPrevFont = ""; // 바로 이전의 폰트
7:     LOGFONT lf;
8:
9:     // LOGFONT 구조체를 초기화한다
10:    lf.lfCharSet = DEFAULT_CHARSET;
11:    strcpy(lf.lfFaceName, "");
12:    // 리스트 박스를 썩 청소한다
13:    m_ctlFontList.ResetContent();
14:    // 디바이스 컨텍스트를 잡아낸다
15:    CClientDC dc (this);
16:    // 폰트 부류를 나열한다
17:    ::EnumFontFamiliesEx((HDC) dc, &lf,
18:    (FONTENUMPROC) EnumFontFamProc, (LPARAM) this, 0);
19:    // 리스트 박스내의 폰트 수를 얻어낸다
20:    iCount = m_ctlFontList.GetCount();
21:    // 리스트 박스를 끝부터 처음까지 순환하면서
22:    // 중복된 엔트리를 찾아 제거한다
23:    for (iCurCount = iCount; iCurCount > 0; iCurCount--)
24:    {
25:        // 현재 폰트 이름을 얻는다
26:        m_ctlFontList.GetText((iCurCount - 1), strCurFont);
27:        // 이전의 폰트이름과 같은고?
28:        if (strCurFont == strPrevFont)
29:        {
30:            // 그렇소만, 지우겠소
31:            m_ctlFontList.DeleteString((iCurCount - 1));
32:        }
33:        // 이전 폰트 이름을 현재 폰트 이름으로 설정한다
34:        strPrevFont = strCurFont;
35:    }
36: }
```

리스트 후반에 나오는 for 루프를 주목하자. 리스트의 끝부터 처음으로 순환하고 있기 때문에 리스트 박스 안의 라인을 뛰어넘을 위험을 막기 위해 루프 카운터를 조작

할 걱정을 하지 않아도 무난하게 현재 엔트리를 삭제할 수 있게 된다. 애플리케이션을 컴파일하고 실행하면, 더 이상 중복된 이름이 나타나지 않을 것이다.

■ 폰트로 출력할 텍스트를 설정하자

폰트를 사용자에게 보여주기 전에, 디스플레이 영역에 나타낼 텍스트가 하나쯤은 있어야 하지 않을까? 다이얼로그 윈도우의 위쪽 가까이 있는 에디트 박스가 바로 그런 용도로 준비된 것이다. 이제는 에디트 박스가 제구실을 할 수 있도록 하자.

1. OnInitDialog() 함수를 열고 [리스트 7.6]과 같이 에디트 박스를 초기화하는 코드를 추가한다.

리스트 7.6 고쳐진 OnInitDialog() 함수

```
1: BOOL CDay7Dlg::OnInitDialog()
2: {
3:     CDialog::OnInitDialog();
4:     .
5:     .
6:     .
7:     // TODO: Add extra initialization here
8:     .
9:     /////////////////
10:    // 새로 넣을 코드가 여기서부터 시작된다
11:    ///////////////
12:    // 리스트 박스를 폰트로 채운다
13:    FillFontList();
14:
15:    // 입력될 텍스트를 초기화한다
16:    m_strSampText = "Testing";
17:    // 폰트 샘플 영역에 텍스트를 복사한다
18:    m_strDisplayText = m_strSampText;
19:    // 다이얼로그를 갱신한다
20:    UpdateData(FALSE);
21:
22:    .
23:    /////////////////
24:    // 새로 넣을 코드는 여기서 끝난다
25:    ///////////////
26:
27:    return TRUE; // return TRUE unless you set the focus to a control
28: }
```

2. 클래스워저드를 사용해서 IDC_ESAMPTEXT 에디트 박스 컨트롤의 EN_CHANGE 이벤트 메시지에 대한 함수를 추가한다.
3. 방금 추가한 함수에 [리스트 7.7]과 같은 코드를 입력하자.

리스트 7.7 OnChangeEsamptext() 함수

```

1: void CDay7Dlg::OnChangeEsamptext()
2: {
3:     // TODO: If this is a RICHEDIT control, the control will not
4:     // send this notification unless you override the
5:     // → CDialog::OnInitialUpdate()
6:     // function and call CRichEditCrtl().SetEventMask()
7:     // with the EN_CHANGE flag ORed into the mask.
8:
9:     // TODO: Add your control notification handler code here
10:    ///////////////
11:    // 새로 넣을 코드가 여기서부터 시작된다
12:    ///////////////
13:
14:    // 다이얼로그 상의 컨트롤에 있는 내용을 둘려 둔 변수로 읽어온다
15:    UpdateData(TRUE);
16:
17:    // 폰트 샘플영역의 텍스트 변수로 현재 텍스트를 복사한다
18:    m_strDisplayText = m_strSampText;
19:
20:    // 다이얼로그를 갱신한다
21:    UpdateData(FALSE);
22:
23:    ///////////////
24:    // 새로 넣을 코드는 여기서 끝난다
25:    ///////////////
26: }
```

애플리케이션을 컴파일하고 실행하면, 에디트 박스에 텍스트를 써넣을 때마다 아래쪽의 그룹 박스 안에 표시되는 내용이 바뀌는 것을 확인할 수 있을 것이다.

표시할 폰트를 선택해 주자

디스플레이 영역에 표시할 폰트를 바꾸기 위해서는 일단 다이얼로그 클래스 안에 CFont 타입의 멤버 변수를 추가할 필요가 있다.

1. 프로젝트 워크스페이스의 클래스뷰를 선택하고 CDay7Dlg 클래스에서 오른쪽 클릭한다. 팝업 문맥 메뉴에서 Add Member Variable을 선택한다.

- TRY IT**
- 변수 타입으로 CFont를, 변수 이름으로 m_fSampFont를 입력하고, 액세스 지정자로는 Private를 선택한다. OK 버튼을 클릭해서ダイ얼로그 박스를 닫으면 변수 추가가 끝난다.

선택된 폰트를 사용하는 코드를 추가할 때는 컨트롤의 멤버로 넣지 않도록 하자. 왜 이렇게 하는지는 좀더 읽다 보면 확실히 이해할 수 있을 것이다. 선택된 폰트를 표시하고 사용하는 함수를 추가하기 위해 다음의 단계를 따르자.

- 프로젝트 워크스페이스의 클래스뷰를 선택하고 CDay7Dlg 클래스에서 오른쪽 클릭한다. 팝업 메뉴에서 Add Member Function을 선택한다.
- 함수 타입에는 void, 함수 선언에는 SetFont()를 입력하고, 액세스 지정자로는 Private를 선택한다. OK 버튼을 눌러서ダイ얼로그를 닫으면 함수 추가가 끝난다.
- [리스트 7.8]과 같이, 추가한 함수에 코드를 입력하자.

리스트 7.8 SetMyFont() 함수

```

1: void CDay7Dlg::SetFont()
2: {
3:     CRect rRect;          // 디스플레이 영역의 사각형
4:     int iHeight;          // 디스플레이 영역의 높이
5:
6:     // 선택된 것이 있는가?
7:     if (m_strFontName != "")
8:     {
9:         // 폰트 샘플 디스플레이 영역의 디멘션을 얻는다
10:        m_ctlDisplayText.GetWindowRect(&rRect);
11:        // 이 영역의 높이를 계산한다
12:        iHeight = rRect.top - rRect.bottom;
13:        // 높이를 양수로 맞춘다.
14:        if (iHeight < 0)
15:            iHeight = 0 - iHeight;
16:        // 현재 폰트를 떼어낸다
17:        m_fSampFont.Detach();
18:        // 사용될 폰트를 생성한다
19:        m_fSampFont.CreateFont((iHeight - 5), 0, 0, 0, FW_NORMAL,
20:                             0, 0, 0, DEFAULT_CHARSET, OUT_CHARACTER_PRECIS,
21:                             CLIP_CHARACTER_PRECIS, DEFAULT_QUALITY, DEFAULT_PITCH :
22:                             FF_DONTCARE, m_strFontName);
23:
24:        // 텍스트를 표시하는 디스플레이 영역의 폰트를 설정한다
25:        m_ctlDisplayText.SetFont(&m_fSampFont);
26:    }
27: }
```

4. 클래스위저드를 사용해서 IDC_LFONTS 리스트 박스의 LBN_SELCHANGE 이벤트 메시지에 대한 함수를 추가하고, [리스트 7.9]와 같이 만든다.

리스트 7.9 OnSelchangeLfonts() 함수

```

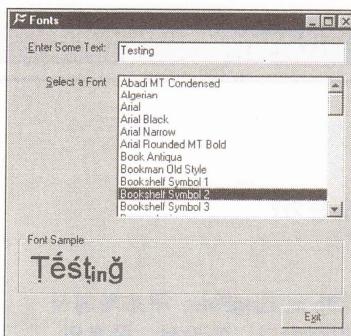
1: void CDay7Dlg::OnSelchangeLfonts()
2: {
3:     // TODO: Add your control notification handler code here
4:
5:     /////////////////////
6:     // 새로 넣을 코드가 여기서부터 시작된다
7:     /////////////////////
8:
9:     // 다이얼로그 컨트롤에 물려 둔 변수를 갱신한다
10:    UpdateData(TRUE);
11:
12:    // 샘플 텍스트를 위한 폰트를 설정한다
13:    SetMyFont();
14:
15:    /////////////////////
16:    // 새로 넣을 코드는 여기서 끝난다
17:    /////////////////////
18: }
```

SetMyFont() 함수에서는 우선 선택된 폰트가 있는지, 없는지를 점검했다. 다음 폰트를 사용해서 출력할 텍스트를 나타낼 공간인 정적 텍스트 컨트롤의 사각형 영역을 얻어내었는데, 표시될 영역보다 작은 폰트 높이를 설정하기 위해서이다. 이 정적 텍스트 컨트롤의 높이를 계산하고 이 값이 양수인가를 점검한 후, 선택된 폰트를 생성하고 정적 텍스트 컨트롤로 하여금 이 폰트를 사용하라고 알려주었다.

OnSelchangeLfonts() 함수에서는 컨트롤에 물려둔 변수를 모두 현재의 컨트롤 내용으로 갱신한 다음, SetMyFont() 함수를 호출해서 선택된 폰트를 사용할 수 있도록 하였다. 이제 여러분의 완성된 애플리케이션을 컴파일하고 실행하면, 시스템에서 사용할 수 있는 폰트를 선택한 다음, 에디트 박스에 텍스트를 입력하면서 아래 부분에 선택된 현재의 폰트로 출력되는 텍스트를 눈으로 확인할 수 있을 것이다.

그림 7.3 →

선택된 폰트를 가지고 텍스트를 표시한다.

**요약**

이번 장에서 배운 것은 비주얼 C++ 애플리케이션에서의 폰트 사용이다. 시스템에 설치되어 있는 사용 가능한 폰트 리스트를 얻어내고 디스플레이 개체에 사용할 용도의 폰트를 생성하는 방법을 배웠으며, 윈도우 운영체제로부터 폰트의 리스트를 얻어내는데 필요한 콜백 함수를 만들고 사용하는 방법까지 학습하였다. 아울러, 폰트 리스트를 요구할 때 콜백 함수에 넘겨준 윈도우 포인터를 사용해서 콜백 함수를 제어하는 방법을 끝으로 7장의 공부를 맺었다.

Q&A

CreateFont() 함수의 엄청나게 많은 인수를 외울려다가는 머리가 세어지겠군요. 이 함수 대신 사용할 수 있는 대안이 있나요?

CreateFont()에 넘겨지는 정보를 모두 설정해야 하지만, 물론 존재합니다. LOGFONT라고 불리는 구조체는 CreateFont() 함수에 넘겨지는 똑같은 속성을 멤버로 가지고 있습니다. 이 구조체의 인스턴스를 메모리에 선언한 다음, 각 속성을 디폴트값으로 초기화하시고, CreateFontIndirect() 함수에 넘기십시오. 상당히 많은 폰트 변화가 필요하다면 이 방법이 편합니다. 왜냐하면, 폰트 생성에 쓰이는 정보를 가진 구조체를 하나만 가지고 사용하니까요. 이 구조체의 멤버를 수정하기만 하면 되잖습니까.

방금 설명한 방법을 우리가 만든 애플리케이션에 적용하려면, LOGFONT 구조체의 인스턴스를 선언하고 모든 속성을 디폴트값으로 초기화하는 과정을 SetMyFont()을 호출하기 전에 해두어야 합니다. SetMyFont() 함수에서는 [리스트 7.10]과 같이 해줍시다.

리스트 7.10 수정된 SetMyFont() 함수

```

1: void CDay7Dlg::SetMyFont()
2: {
3:
4:     // 폰트가 선택된 것이 있는가?
5:     if (m_strFontName != "")
6:     {
7:         // 폰트의 크기는 이미 m_lLogFont 구조체에서
8:         // 초기화되었다고 가정한다. 따라서, 폰트의
9:         // 이름만 설정하면 된다.
10:        tcscpy(m_lLogFont.lfFaceName, m_strFontName);
11:        // 사용될 폰트를 생성한다
12:        m_fSampFont.CreateFontIndirect(&m_lLogFont);
13:
14:        // 디스플레이 영역에 폰트를 설정한다
15:        m_ctlDisplayText.SetFont(&m_fSampFont);
16:    }
17: }
```

➊ 제가 얻어낸 폰트 리스트에서 트루타입(TrueType) 폰트만 포함되도록 제한할 수 없을까요?

➋ 여러분이 만든 콜백 함수에서 nFontType 인수를 체크하여 폰트의 타입을 결정합니다. 예를 들어, 폰트 리스트에서 트루타입 폰트만 포함되도록 하려면, 콜백 함수 안에서 nFontType 인수를 TRUETYPE_FONTTYPE 상수로 매스킹하고, 그 결과값이 TRUETYPE_FONTTYPE인지만 알아보면 되지요. 바로 다음과 같아요.

```

int CALLBACK EnumFontFamProc(LPENUMLOGFONT lpef,
LPNEWTEXTMETRIC lpntm, DWORD nFontType, long lParam)
{
    // 마지막 인수를 다이얼로그 원도우의 포인터로 캐스팅한다.
    CDay7Dlg* pWnd = (CDay7Dlg*) lParam;

    // 트루타입 폰트만이 폰트리스트 포함되도록 한다
    ✓ if ((nFontType & TRUETYPE_FONTTYPE) == TRUETYPE_FONTTYPE)
    {
        // 리스트 박스에 폰트 이름을 추가한다
        pWnd->m_ctlFontList.AddString(
            lpef->elfLogFont.lfFaceName);
    }
    // 폰트 나열을 계속하기 위해 1을 반환한다
    return 1;
}
```

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 있고, 정답 보고 풀지 말아라.

퀴즈

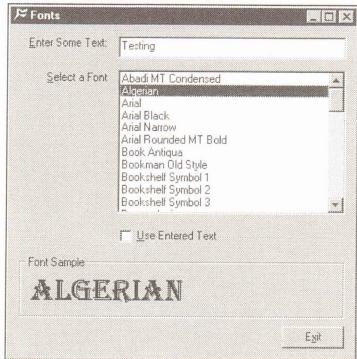
1. 텍스트에 밑줄을 그으려면 어떻게 할까?
2. 텍스트를 뒤집어서 출력하려면 어떻게 할까?
3. 운영체제에 의해 EnumFontFamProc() 콜백 함수가 얼마나 자주 호출되는가?

연습문제

1. 체크 박스를 추가해서, [그림 7.4]와 같이 입력된 텍스트를 사용해서 폰트를 표시하는 모드와 폰트 이름을 사용해서 폰트를 표시하는 모드 사이를 전환할 수 있도록 해보자.

그림 7.4 →

폰트 이름을 해당 폰트로 출력한다.



2. 체크 박스를 추가해서 [그림 7.5]와 같이 폰트 샘플을 이탤릭체로 표시할 수 있도록 해보자.

그림 7.5 →

선택된 폰트를 이탤릭으로 표시한다.

