# 윈도우 NT 서비스 어플리케이션의 제작

윈도우 NT 서버를 기반으로 한 클라이언트/서버 어플리케이션을 개발하다 보면, 컴퓨터에 관리자가 로그온하지 않은 상황에서도 서버 어플리케이션을 동작할 필요가 있다. 이럴 때에 사용되는 것인 NT 서비스 어플리케이션이다. 시작 프로그램 그룹에 서버 어플리케이션을 등록한 경우에는 사용자의 로그온이 있어야만 시작이 되지만, 서비스 어플리케이션이 윈도우 NT 가 기동만 하면 로그온 작업 여부와 관계없이 사용이 가능하다.
델파이 4 에서는 이러한 윈도우 NT 서비스 어플리케이션을 쉽게 작성할 수 있는 클래스 들과 위저드가 제공된다.
이번 장에서는 델파이 4 에서 제공되는 위저드를 이용하여 윈도우 NT 서비스 어플리케이션을 작성하는 방법에 대해서 알아볼 것이다.

## 서비스 어플리케이션 위저드

Win32 서비스를 구현하는 어플리케이션을 제작하려면, File|New 메뉴를 선택하고 객체 저장소의 New 탭에서 Service Application 아이템을 더블 클릭하면 된다. 이렇게 하면 TServiceApplication 클래스 형의 Application 변수가 프로젝트에 추가될 것이다.
일단 이렇게 서비스 어플리케이션을 생성하면 서비스에 해당되는 윈도우가 나타날 것이다. 서비스 자체는 TService 클래스를 상속하여 구현하게 되는데, 오브젝트 인스펙터에서 이 객체의 프로퍼티를 설정하고, 이벤트 핸들러를 작성하면 된다.
서비스 어플리케이션에 추가적인 서비스를 지원하게 하려면 File|New 메뉴의 New 탭에서 Service 아이템을 더블 클릭하여 하나의 어플리케이션에 여러 개의 서비스를 지원하게 할 수도 있다.

## 서비스 관련 클래스

서비스 관련 클래스로 델파이에서 제공되는 것으로는 TServiceApplication, TService, TDependency 클래스가 있다.

- TServiceApplication

TServiceApplication 클래스는 윈도우 NT 서비스 어플리케이션을 캡슐화한 클래스이다. 이 클래스는 서비스 어플리케이션에 가장 기본적인 기능을 제공한다. 각각의 서비스 프로젝트는 자동으로 어플리케이션의 인스턴스로 TServiceApplication 클래스 형인 Application 변수를 선언하게 한다.

TServiceApplication 클래스는 윈도우 NT 서비스를 캡슐화한 TService 객체 들을 포함하는데, 서비스 어플리케이션 객체에는 서비스 객체를 생성, 설치, 등록, 디스패치, 제거하는 메소드를 제공한다.

- TService

TService 클래스는 윈도우 NT 서비스 어플리케이션에서 NT 서비스를 캡슐화한다. Win32 서비스는 SCM(Service Control Manager)에 의해 접근할 수 있으며, 시스템이 시작할 때 자동으로 서비스를 개시하거나, 사용자가 서비스 제어판 애플릿을 이용하여 수동으로 시작하게 할 수 있으며, Win32 기반의 어플리케이션에서 서비스 함수를 사용할 때 시작하게 할 수도 있다.

- TDependency

TDependency 객체는 새로운 서비스나 TService 객체에 의해 구현된 서비스가 시작되기 전에 시작해야 하는 ordering 그룹을 적재하는 역할을 한다.
Dependency 객체 들의 컬렉션은 TDependencies 객체이며, TService 클래스는 Dependencies 와 LoadGroup 프로퍼티를 이용하여 이 컬렉션을 이용할 수 있다. 각각의 dependency 객체의 Name 프로퍼티는 서비스 또는 ordering 그룹의 이름을 결정하며, IsGroup 프로퍼티를 이용하여 이것이 서비스인지 그룹인지 나타내게 된다.

## 서비스 쓰레드

각각의 서비스는 자신의 쓰레드를 가지고 있다. 이런 쓰레드는 TServiceThread 클래스에서 상속받은 객체인데, 서비스 어플리케이션이 하나 이상의 서비스를 구현한 경우에는 서비스가 쓰레드에 안전하도록 작성해야 한다는 것을 잊어서는 안된다.
TServiceThread 클래스는 TService 클래스의 OnExecute 이벤트 핸들러를 작성함으로써 구현될 수 있게 디자인 되었다. 그러므로, 서비스 쓰레드는 새로운 요구를 처리하기 전에 서비스의 OnStart 와 OnExecute 이벤트 핸들러를 호출하는 루프를 포함한 자신의 Execute 메소드를 가지고 있다.
서비스는 처리하는데 비교적 오랜 시간이 걸리며, 동시에 여러 요구를 하나 이상의 클라이언트에서 받을 가능성이 많기 때문에, 이럴 때에는 각각의 요구에 대해서 TThread 클래스에서 상속받은 새로운 쓰레드를 사용하는 것이 효과적이다. 그리고, 이런 경우에는 새로운 쓰레드의 Execute 메소드를 구현해야 한다.

## 서비스 name 프로퍼티

TService, TDependency 클래스와 같이 서비스 어플리케이션을 작성하는데 제공되는 VCL 클래스를 이용할 때에는 name 프로퍼티에 대해서 잘 알아둘 필요가 있다.
서비스는 Service Start names 라고 하는 사용자 이름을 가지고 있는데, 여기에는 패스워드가 연관되어 있다. 이 이름은 서비스 관리자에 표시되며, 에디터 윈도우나 서비스의 실제 이름으로 사용된다.
그에 비해, Dependency 는 서비스일 수도 있고, 그룹으로 적재될 수도 있다. 문제는 dependency 도 이름과 display 이름을 가진다는 점이다. 그 밖에도 서비스 객체 역시 TComponent 클래스에서 상속받은 객체이기 때문에 Name 프로퍼티를 가지고 있다.
이들의 차이점에 대해서 간단히 정리해 보도록 하자.

- TDependency 클래스의 DisplayName 프로퍼티

TDependency 의 DisplayName 프로퍼티는 프로퍼티 에디터 윈도우에 표시되는 dependent 서비스의 이름을 나타낸다. TDependency 클래스의 Name 프로퍼티와 거의 동일한 역할을 한다.

- TService 클래스의 Name 프로퍼티

TService 의 Name 프로퍼티는 TComponent 에서 상속된 프로퍼티로, 컴포넌트의 이름이 되는 동시에 서비스의 이름이 된다. 만약 dependency 가 서비스이면 TDependency 클래스의 Name, DisplayName 프로퍼티와 같은 역할을 한다. 주의할 것은 TService 클래스에도 DisplayName 프로퍼티가 있는데, 이 프로퍼티는 완전히 다른 것이므로 주의하기 바란다.

## 서비스 관련 클래스의 주요 프로퍼티와 메소드

예제 어플리케이션을 작성하기에 앞서 윈도우 NT 서비스 어플리케이션을 작성하기 쉽도록 제공되는 델파이 클래스의 주요 프로퍼티와 메소드에 대해서 알아보도록 하자.

- TServiceApplication 클래스

TServiceApplication 에서 중요한 프로퍼티로는 ServiceCount, Title 이 있다.
ServiceCount 프로퍼티는 서비스 어플리케이션의 서비스 수를 나타낸다. 각각의 서비스는 TService 인스턴스이며 자신의 서비스 쓰레드를 가지고 있다. Title 프로퍼티는 서비스 어

플리케이션의 이름을 가리키게 된다.
또한 중요한 메소드로는 CreateForm, Initialize, Run 메소드가 있다.
CreateForm 메소드는 서비스 어플리케이션에 포함된 각 서비스에 대한 TService 객체를 생성하는 메소드로, 서비스 객체가 추가될 때마다 자동으로 생성된다. Initialize 메소드는 어플리케이션과 관련된 폼을 숨기기 위한 초기 설정을 한다. 서비스 어플리케이션은 거의 interactive 할 필요가 없기 때문에 폼을 보여줄 필요가 없다. 만약 폼이 어플리케이션에 추가될 경우 Initialize 메소드는 시작할 때 이를 숨기게 된다. Run 메소드는 서비스를 설치, 등록하고 컴포넌트 리스트에 서비스를 추가하며 서비스 어플리케이션의 쓰레드를 생성하는 역할을 한다.

- TService 클래스

1. 주요 프로퍼티

AllowPause, AllowStop 프로퍼티는 서비스의 클라이언트가 서비스를 일시 중단하거나 중단할 수 있는지 여부를 결정한다. 이 값이 True 이면 OnPause, OnContinue 이벤트는 서비스가 일시 중단되거나 다시 시작될 때 발생하며, OnStop 이벤트는 서비스가 중지되기 전에 발생한다.
Dependencies 프로퍼티는 TDependency 객체들의 컨테이너로 사용되는 프로퍼티이다. 이 프로퍼티에는 서비스가 시작되기 전에 시작하는 dependent 서비스나 load ordering 그룹이 지정된다. 서비스에서의 Dependency 가 의미하는 바는 이 서비스가 dependency 로 지정된 서비스가 시작되지 않는 한 실행되지 않는다는 의미이다. 그룹에서의 Dependency 가 의미하는 바는 그룹으로 지정된 서비스들 중에서 최소한 하나의 멤버가 시작되어야 실행된다는 의미이다.
DisplayName 프로퍼티는 SCM 에 나타나는 서비스의 이름을 나타낸다. 여기에 대해서는 앞에서 다룬바 있으므로 자세한 설명은 생략한다.
에러 처리를 위한 프로퍼티로는 ErrCode 와 ErrorSeverity 프로퍼티가 있다. ErrCode 프로퍼티는 서비스가 시작되거나 중단되는 동안 발생하는 에러의 코드가 어떤 것인지를 나타낸다. 이 프로퍼티가 정의되지 않은 경우에는 Win32ErrCode 프로퍼티가 대신 사용된다.
ErrorSeverity 프로퍼티는 서비스가 기동될 때 실패할 경우에 그 심각한 정도를 나타내는 프로퍼티로 다음과 같은 값들을 가질 수 있다.

| 값 | 의 미 |
|---|---|
| esIgnore | 에러를 나타내지만 무시한다. |
| esNormal | 에러가 발생하고, 메시지가 표시되지만 실행은 중지되지 않는다. |
| esSevere | 에러가 발생하지만, 마지막(last-known-good) 환경 설정을 사용하면 실행될 |

| | |
|---|---|
| | 수 있는 경우 |
| esCritical | 에러가 발생하고, 과거 설정으로 시작을 시도하되 이 역시 실패하면 실행이 중단된다. |

ServiceType 프로퍼티는 서비스의 종류를 나타낸다. 그 값으로는 stWin32(Win32 서비스, 디폴트), stDevice (디바이스 드라이버), stFileSystem (파일 시스템 드라이버) 일 수 있다.
ServiceStartName 프로퍼티는 서비스를 시작할 때 사용되는 이름을 나타낸다. 만약 ServiceType 프로퍼티의 값이 stWin32 이면 ServiceStartName 프로퍼티의 의미는 'DomainName₩UserName' 형태의 계정 이름이다. 만약 계정이 built-in 도메인에 속해 있는 경우에는 '.₩UserName' 형태로 지정할 수도 있다. 서비스 어플리케이션이 하나 이상의 서비스를 포함할 때에는 이 프로퍼티와 Password 프로퍼티를 비워둔다.
ServiceType 프로퍼티의 값이 stWin32 가 아니면 ServiceStartName 프로퍼티 값은 디바이스 드라이버를 로드하기 위해 사용되는 입출력 시스템의 드라이버 객체의 이름인 '₩FileSytems₩Rdr 또는 ₩Driver₩Xns' 형태를 가진다.
Password 프로퍼티는 ServiceType 프로퍼티의 값이 stWin32 로 지정되었을 때 ServiceStartName 프로퍼티에 지정된 계정의 패스워드를 나타낸다. 이 프로퍼티의 값이 NULL 이거나 빈 문자열을 가리킬 경우 서비스에는 패스워드가 없는 것이다.
StartType 프로퍼티는 서비스가 어떻게 시작할 것인지 지정한다. 다음과 같은 값을 가질 수 있다.

| 값 | 시작되는 방법 | 비 고 |
|---|---|---|
| stBoot | 운영체제 로더에 의해 시작된다. | ServiceType 이 stWin32 가 아닌 경우 |
| stSystem | 부트 시스템이 초기화된 후에 시작된다. | ServiceType 이 stWin32 가 아닌 경우 |
| stAuto | 시스템이 시작될 때 자동으로 시작된다. | |
| stManual | 프로세스가 StartService API 함수를 호출할 때 시작된다. | |
| stDisabled | 서비스를 관리자가 수동으로 시작해야 한다. | |

Interactive 프로퍼티는 서비스가 윈도우 데스크 탑과 상호작용할 수 있는지 여부를 나타낸다. Interactive 프로퍼티는 ServiceType 프로퍼티가 stWin32 인 경우에만 효과가 있다.
LoadGroup 프로퍼티는 서비스를 포함한 load ordering 그룹의 이름을 나타낸다. 이 프로퍼티는 load ordering 그룹에서 dependecy 를 가진 다른 서비스들에 의해 사용된다. 레지스트리는 load ordering 그룹을 HKLM₩System₩CurrentControlSet₩Control₩ServiceGroupOrder 키에서 확인할 수 있다.
서비스들과 서비스 그룹이 같은 name space 를 공유하기 때문에, 그룹 이름의 접두어는 반드시 SC_GROUP_IDENTIFIER 을 사용해서 서비스 이름과 구별된다. 이 접두어는

TDependency 클래스의 IsGroup 프로퍼티가 True 이면 자동으로 붙는다.
Param 프로퍼티는 서비스에 대한 파라미터 들을 담은 인덱스된 프로퍼티이다. 이들 파라미터는 SCM 의 시작 파라미터 입력 윈도우에 그대로 사용된다. ParamCount 프로퍼티에는 이렇게 설정된 파라미터의 수가 지정된다.
ServiceThread 프로퍼티는 서비스에 대한 쓰레드로 사용되는 TServiceThread 클래스 인스턴스를 나타낸다.
Status 프로퍼티는 서비스의 현재 상태를 나타내는데, 이 상태 정보는 SCM 에 표시된다. 다음과 같은 값들을 가질 수 있다.

| 값 | 의 미 |
|---|---|
| csStopped | 서비스가 중단되어 있다. |
| csStartPending | 서비스가 시작하려 하고 있다. |
| csStopPending | 서비스가 중단되고 있다. |
| csRunning | 서비스가 실행되고 있다. |
| csContinuePending | 서비스가 continue 되려 하고 있다. |
| csPausePending | 서비스 pause 되려 하고 있다. |
| csPaused | 서비스가 일시 중단되었다. |

Terminated 프로퍼티는 현재 실행되고 있는 서비스의 쓰레드가 중단되었는지 여부를 가리킨다.

2. 주요 메소드

GetServiceController 메소드는 서비스에 대한 핸들러의 포인터를 반환한다. 이 함수는 직접 호출할 필요는 없고, 서비스 객체의 메인 함수가 이를 자동으로 호출한다.
서비스 어플리케이션에서 각각의 서비스는 서비스에 대한 메인 함수의 포인터를 가지고 있다. 요구에 의해 서비스가 시작되면 서비스의 메인 쓰레드는 RegisterServiceCtrlHandler 함수를 호출하여 이 컨트롤 핸들러 함수를 등록하게 되며, 이 함수에 대한 포인터를 반환한다. 이 함수는 클래스 메소드가 아니기 때문에 서비스 어플리케이션 위저드는 논-멤버 핸들러를 생성하게 되는 것이다.
LogMessage 메소드는 에러 메시지를 이벤트 로그에 전송한다. 이때 디폴트로 EventType 파라미터의 값은 EVENTLOG_ERROR_TYPE, ID 와 Category 파라미터는 0 으로 지정되어 있다. 예를 들어, 서비스가 시스템이 부트될 때 로드되지 못하면 에러 이벤트를 로그에 기록한다. EventType 파라미터의 값으로는 다음과 같은 값들을 지정할 수 있다.

| 값 | 의 미 |
|---|---|
| EVENTLOG_ERROR_TYPE | Error event |
| EVENTLOG_WARNING_TYPE | Warning event |
| EVENTLOG_INFORMATION_TYPE | Information event |
| EVENTLOG_AUDIT_SUCCESS | Success Audit event |
| EVENTLOG_AUDIT_FAILURE | Failure Audit event |

Category 파라미터는 이벤트 카테고리를 지정하는데, 특정 소스에 대한 정보를 가진다. ID 파라미터는 이벤트 ID 를 지정하는데, 이것은 이벤트 소스와 연관된 메시지 파일에서 엔트리로 사용되는 이벤트이다.
ReportStatus 메소드는 현재의 상태 정보를 전송하는 역할을 한다.

3. 주요 이벤트

BeforeInstall 이벤트는 서비스가 처음 등록되기 전에 발생하며, AfterInstall 이벤트 핸드러는 서비스가 등록된 이후에 발생한다. 마찬가지로 BeforeUninstall 이벤트는 서비스가 제거되기 직전에, AfterUninstall 이벤트 핸들러는 서비스가 제거된 직후에 발생한다.
OnStart 이벤트 핸들러는 서비스를 초기화할 때 사용되는데, 예를 들어 서비스 요구가 분리된 쓰레드로 처리된다면 쓰레드에 대한 여러가지 처리 사항은 이 이벤트 핸들러에서 처리하고 Started 파라미터를 True 로 설정하면 서비스를 시작하게 된다. OnStop 이벤트 핸들러는 서비스가 중단될 때 발생하며 Stopped 파라미터를 True 로 설정하면 서비스가 중단된다.
OnShutdown 이벤트 핸들러는 서비스가 shutdown 되기 직전에 발생하며, 서비스 어플리케이션은 이 이벤트가 발생한 직후에 종료된다.
OnPause 이벤트 핸들러에서는 서비스가 일시 중단될 때의 처리를 담당하며 Paused 파라미터를 True 로 설정하면 서비스가 일시 중지된다. OnContinue 이벤트 핸들러는 SCM 이 일시 중지된 서비스를 다시 시작할 때 발생하는데 Continued 파라미터를 True 로 설정하면 서비스가 재시작된다.
OnExecute 이벤트 핸들러는 개별 쓰레드가 OnStart 이벤트 핸들러에서 요구될 때 서비스를 직접 구현하는 핸들러로 사용된다. OnExecute 이벤트 핸들러가 종료되면, 서비스 쓰레드도 종료된다. 대부분의 OnExecute 이벤트 핸들러는 서비스 쓰레드의 ProcessRequest 메소드를 호출하는 루프를 이용하여 다른 서비스에 대한 요구가 중단되지 않도록 배려하도록 한다.

## 소켓을 이용한 서비스 어플리케이션 예제

그러면, 서비스 어플리케이션 위저드를 이용한 서비스 어플리케이션 예제를 하나 작성해보자.  작성할 서비스 어플리케이션은 TServerSocket 컴포넌트를 사용하여 서버의 현재 시간을 접속된 모든 클라이언트 어플리케이션에게 전송하는 일종의 타임 서버이다.
File|New 메뉴를 선택하고 New 탭에서 Service Application 아이템을 더블 클릭하면 새로운 프로젝트가 생성되면서, 데이터 모듈과 비슷한 윈도우가 하나 열릴 것이다.
여기에 다음과 같이 TTimer 와 TServerSocket 컴포넌트를 떨어뜨리도록 한다.


그리고, 프로젝트와 유닛 파일을 적당한 이름으로 저장하도록 하자.  이렇게 하면 이미 서비스 어플리케이션 위저드에 의해 다음과 같은 코드가 자동으로 생성되었을 것이다.

```
program ExamSvr1;

uses
  SvcMgr,
  U_ExamSvr1 in 'U_ExamSvr1.pas' {Service1: TService};

{$R *.RES}

begin
  Application.Initialize;
  Application.CreateForm(TService1, Service1);
  Application.Run;
end.

unit U_ExamSvr1;

interface
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, SvcMgr, Dialogs,
```

```
  ScktComp, ExtCtrls;

type
  TService1 = class(TService)
    ServerSocket1: TServerSocket;
    Timer1: TTimer;
  private
  public
    function GetServiceController: PServiceController; override;
    { Public declarations }
  end;

var
  Service1: TService1;

implementation

{$R *.DFM}

procedure ServiceController(CtrlCode: DWord); stdcall;
begin
  Service1.Controller(CtrlCode);
end;

function TService1.GetServiceController: PServiceController;
begin
  Result := @ServiceController;
end;

end.
```

앞에서 설명했듯이 TService 클래스의 GetServiceController 메소드는 이런 방법으로 컨트롤러 메소드의 포인터를 반환하도록 구현되어 있다.
우리가 작성할 서비스 어플리케이션은 서비스가 시작될 때 서버 소켓을 활성화시키고, 1 초에 한번씩 서버 소켓에 연결된 클라이언트에게 현재 시간을 전송하는 것이다.
서비스 어플리케이션을 작성할 때 핵심이 되는 것은 Service1 객체의 OnExecute 이벤트

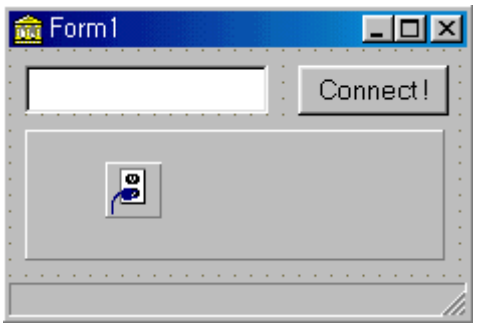
핸들러를 작성하는 것이다. 이 이벤트 핸들러가 서비스가 시작되면 실제로 실행되는 부분이다. 여기에서 서버 소켓을 다음과 같이 활성화하고, 다른 서비스가 계속 실행될 수 있도록 ServiceThread 의 ProcessRequest 메소드를 호출하는 것이 중요하다. 이 타임 서비스는 포트 2001 번을 사용하도록 지정하였다.

```
procedure TService1.Service1Execute(Sender: TService);
begin
  ServerSocket1.Port := 2001;
  ServerSocket1.Active := True;
  while not Terminated do ServiceThread.ProcessRequests(False);
  ServerSocket1.Active := False;
end;
```

이제는 Timer1 의 OnTimer 이벤트 핸들러를 다음과 같이 작성하여, 시스템의 현재 시각을 연결되어 있는 모든 클라이언트 소켓으로 전송하도록 하면 된다.

```
procedure TService1.Timer1Timer(Sender: TObject);
var
  TimeStr: string;
  i: Integer;
begin
  if ServerSocket1.Active then
  begin
    TimeStr := TimeToStr(Now);
    i := 0;
    while True do
    begin
      try
        ServerSocket1.Socket.Connections[i].SendText(TimeStr);
        Inc(i)
      except
        Break;
      end;
    end;
  end;
end;
```

이것으로 서비스 어플리케이션을 작성하였다. 일반적인 다른 서버 어플리케이션을 작성하는 것과 크게 다른 것이 없다는 것을 느낄 수 있을 것이다.
그러면, 이 서비스 어플리케이션을 서버로 사용하는 클라이언트 어플리케이션을 작성하도록 하자. 새로운 어플리케이션을 시작하고 에디트 박스, 버튼, Status bar, 패널과 클라이언트 소켓 컴포넌트를 다음과 같이 폼에 추가하도록 하자.




소켓 컴포넌트의 Port 프로퍼티는 서비스 어플리케이션의 포트 번호와 같이 2001 로 설정하도록 하자. 그리고, 패널 컴포넌트에는 시간을 표시하도록 할 것이므로 적당한 폰트로 설정하기 바란다.
그리고, Button1 의 OnClick 이벤트 핸들러를 다음과 같이 작성하여 서버와 접속하도록 한다.

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  if ClientSocket1.Active then ClientSocket1.Active := False;
  if Length(Edit1.Text) > 0 then
  begin
    ClientSocket1.Address := Edit1.Text;
    ClientSocket1.Active := True;
  end;
end;
```

서버와 접속하면 서버의 호스트 이름을 표시하도록 클라이언트 소켓의 OnConnect 이벤트 핸들러를 다음과 같이 작성하고, 서버 소켓에서 전송하는 시간 정보를 패널에 표시하도록 OnRead 이벤트 핸들러를 다음과 같이 작성한다.

```
procedure TForm1.ClientSocket1Connect(Sender: TObject;
```

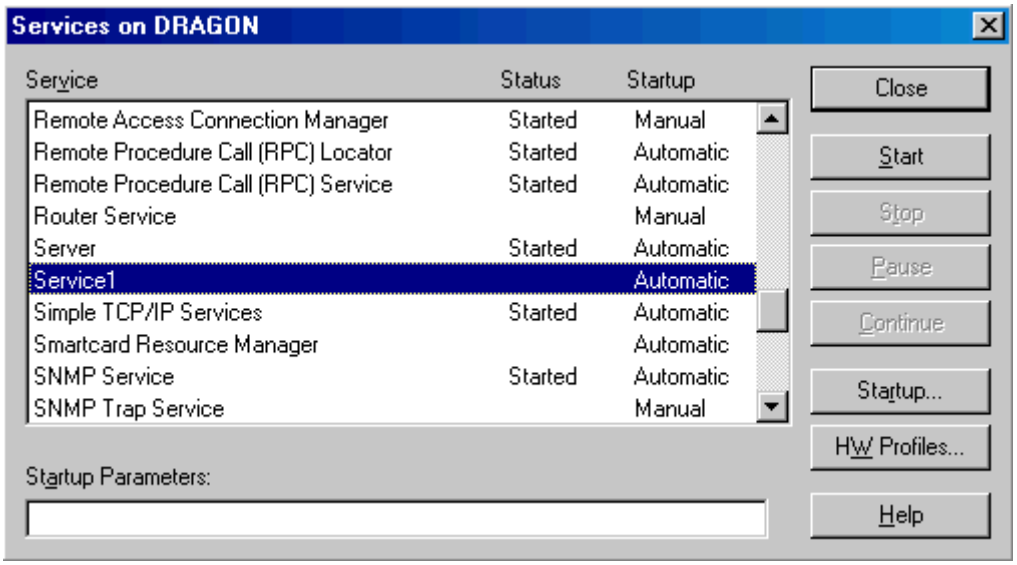
```
  Socket: TCustomWinSocket);
begin
  StatusBar1.SimpleText := 'Connected to ' + Socket.RemoteHost;
end;

procedure TForm1.ClientSocket1Read(Sender: TObject;
  Socket: TCustomWinSocket);
begin
  Panel1.Caption := Socket.ReceiveText;
end;
```

이것으로 클라이언트 어플리케이션도 제작하였다. 그러면 이들을 실행하여 실제 동작하는 화면을 보도록 하자.
먼저, 서비스 어플리케이션 .exe 파일을 윈도우 NT 서버 컴퓨터의 적당한 디렉토리에 위치시키고 다음과 같이 실행하여 서비스를 설치하도록 한다.

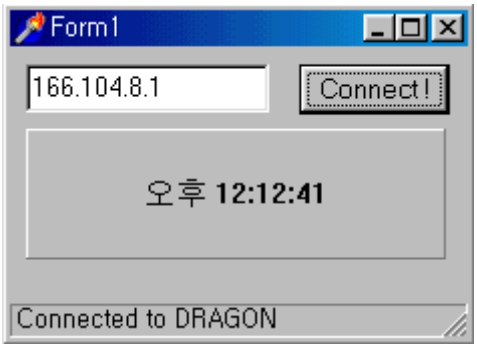
```
ExamSvr1.exe /install
```

참고로, 이렇게 설치한 서비스 어플리케이션을 제거할 때에는 /uninstall 스위치를 사용하면 된다. 그리고, 서비스 컨트롤 관리자를 띄워서 서비스가 설치되었는지 확인하자. 다음 그림과 같이 서버스가 설치된 것을 확인할 수 있을 것이다.




서비스를 실행하려면 Start 버튼을 클릭하면 된다. 참고로, 윈도우 NT 를 다시 시작하면

자동으로 서비스가 시작되지만, 설치한 직후에는 이와 같이 수동으로 서비스를 시작해야 한다.
이제 클라이언트 컴퓨터에서 클라이언트 어플리케이션을 시작하고, 서버의 IP 주소를 지정한 뒤에 'Connect !' 버튼을 클릭하면 다음과 같이 서버의 시간이 패널에 표시되는 것을 관찰할 수 있을 것이다.




## 정 리 (Summary)

이번 장에서는 델파이 4 에서 제공되는 서비스 어플리케이션 위저드를 이용하여 윈도우 NT 서비스 어플리케이션을 작성하는 방법에 대해서 알아보았다. 앞에서도 언급했듯이 윈도우 NT 서비스는 나름대로의 잇점이 많다. 그렇다고, 일반적인 클라이언트/서버 방식으로 작성할 수 있는 프로그램을 억지로 서비스로 만드는 것은 낭비이므로 삼가도록 하자.
클라이언트/서버 방식의 어플리케이션을 개발할 때 윈도우 NT 의 서비스로 서버 어플리케이션을 개발하면 여러가지 문제점을 해결할 수 있는 경우가 많다. 그러므로, NT 서비스 자체의 잇점과 단점을 파악하고 실제로 이것이 현재 프로젝트에 있어서 유용한 것인지를 판단하는 것이 중요할 것이다.