마이크로소프트 트랜잭션 서버의 이용 (II)

이번 장에서는 앞 장에서 설명한 내용을 바탕으로 MTS 를 지원하는 DCOM 객체를 제작하고, 이를 MTS 환경에서 설치하고 실행하는 방법에 대해서 실제 예를 통해 설명할 것이다. 델파이 4 에서 제공하는 위저드의 사용법은 기존의 COM 에 대한 지원 방법과 유사하지만, 더욱 중요하고도 잘 알려지지 않은 것은 서버에서 어떻게 실제로 MTS 컴포넌트와 패키지를 설치할 것이며, 또한 이렇게 설치한 패키지를 클라이언트에서 사용할 수 있는지에 대한 정보이다.

MTS 를 사용한 실제 적용 방법에 대해서 설명하기 전에, 기본적인 환경과 정보에 대해 알 필요가 있다. MTS 를 잘 활용하기 위해서는 기본적으로 서버 컴퓨터에 MTS 서버가 설치되어 있어야 한다. 윈도우 NT 4.0 의 SP3 버전 이상에서는 IIS 와의 연동을 가능하게 해주는 MTS 서버 버전 2.0 을 사용할 수 있으며, 이를 설치하도록 권하고 싶다. 그리고, 기본적으로 클라이언트 역시 DCOM 환경에서 동작하게 되므로 윈도우 95 의 경우 윈도우 95 용 DCOM 을 설치해야 한다. 또한, 31 장에서 언급한 DCOM 환경 설정이 필요한 것은 물론이다. 이 밖에 더 자세한 MTS 의 설치에 대한 의문점, 업데이트된 파일과 기술적인 내용에 대한 정보는 http://www.microsoft.com/com/mts-f.htm 사이트를 방문하여 참고하기 바란다.

그러면, MTS 환경을 이용한 분산 환경의 세계로의 여행을 떠나보자.

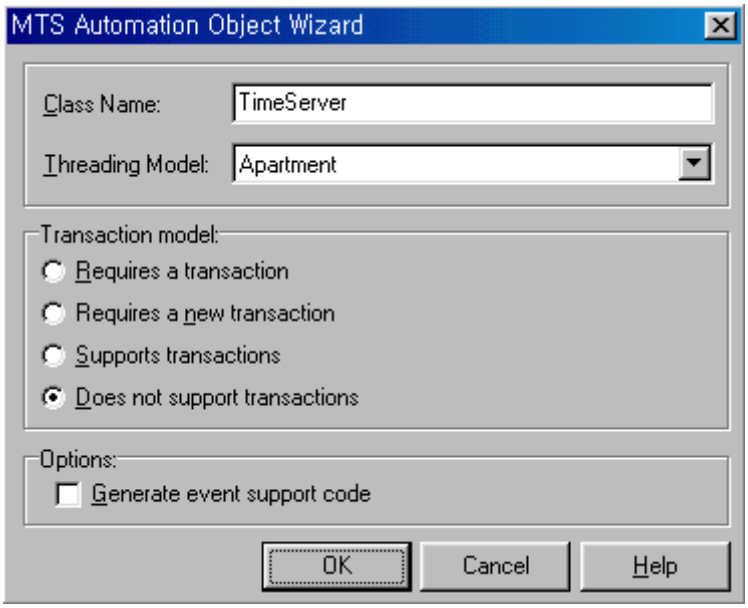
간단한 MTS 객체의 제작

MTS 를 이용한 간단한 서버 객체를 제작하고, 이를 이용한 클라이언트 어플리케이션을 작성하는 방법은 서버 객체와 클라이언트 어플리케이션을 제작하는 것보다 이들을 제대로 설치하는 과정이 훨씬 더 복잡하다. 사실 그렇게 어려운 내용이라고 하기는 어렵지만, 국내에 의외로 MTS 를 이용한 개발 사례가 많지 않고, 이를 활용하는 방법에 대해 언급한 문서가 없어서 필자도 이를 제대로 익히는데 많은 시간을 소비하였다.

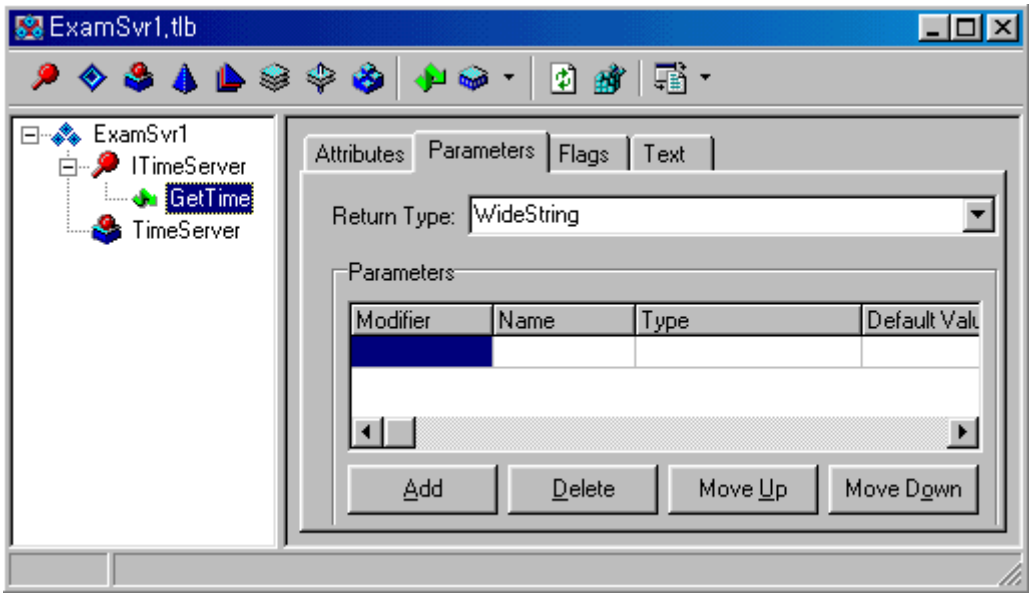
그러면, 실전에 들어가 간단한 MTS 객체를 제작하고 이를 서버와 클라이언트에 제대로 설치하는 방법에 대해서 알아보도록 하자.

먼저 서버 컴포넌트를 작성해야 하는데, 이 과정은 기본적으로 COM 서버 객체를 만드는 과정과 거의 동일하다. 차이점이 있다면 사용하는 위저드가 다르다는 점이다. MTS 객체를 제작하기 위해서는 File|New 메뉴를 선택하고 Multitier 탭에서 MTS Object 아이템을 더블 클릭하여 MTS 자동화 객체 위저드를 실행하도록 하자. 이렇게 하면 다음과 같은 대화상자가 나타나는데, 그 형태가 트랜잭션에 대한 옵션을 선택하는 것을 제외하고는 OLE 자동화 객체 위저드의 대화상자와 거의 같다는 것을 알 수 있을 것이다. 여기에서 다음과 같이 클래스 이름을 적어 넣고, 적당한 옵션을 선택한 후 'OK' 버튼을 클릭한다.

각 옵션의 대한 자세한 설명은 앞 장에서 다룬 바 있으므로 생략하도록 한다.



이렇게 하면, OLE 자동화 서버 위저드와 마찬가지로 타입 라이브러리 에디터가 나타날 것이다. 여기에서 우리가 작성할 MTS 객체 서버는 서버의 시간을 알아낼 수 있는 GetTime 메소드를 지원하는 ITimeServer 인터페이스를 다음과 같이 정의한다.



인터페이스 정의가 끝났으면, 타입 라이브러리 에디터를 종료하면 기본적인 MTS 객체를 생성하기 위한 코드가 자동으로 생성된다. 이들을 적당한 이름으로 저장하도록 하자. 생성된 소스 코드는 다음과 같을 것이다.

```
unit U_ExamSvr1;

interface
uses
  ActiveX, MtsObj, Mtx, ComObj, ExamSvr1_TLB;

type
  TTimeServer = class(TMtsAutoObject, ITimeServer)
  protected
    function GetTime: WideString; safecall;
    { Protected declarations }
  end;

implementation
uses ComServ;

function TTimeServer.GetTime: WideString;
begin
end;

initialization
  TAutoObjectFactory.Create(ComServer, TTimeServer, Class_TimeServer,
    ciMultiInstance, tmApartment);
end.
```

TTimeServer 클래스가 TAutoObject 클래스가 아닌 TMtsAutoObject 클래스에서 상속 받는다는 것을 제외하면 OLE 자동화 서버 위저드에 의해 만들어지는 코드와 완전히 동일한 코드라는 것을 쉽게 알 수 있을 것이다. 그러므로, 필자가 앞에서 소개한 여러 장들의 테크닉 들을 거의 그대로 사용할 수 있다.
구현 부분인 GetTime 메소드만 다음과 같이 구현하여 서버의 시간을 GetTime 메소드를 호출하는 클라이언트에게 문자열 형태로 넘겨주도록 하자. 여기서 TimeToStr 함수를 사용하기 위해 uses 절에 SysUtils.pas 유닛을 추가하도록 한다.

```
function TTimeServer.GetTime: WideString;
begin
```

```
  Result := TimeToStr(Now);
end;
```

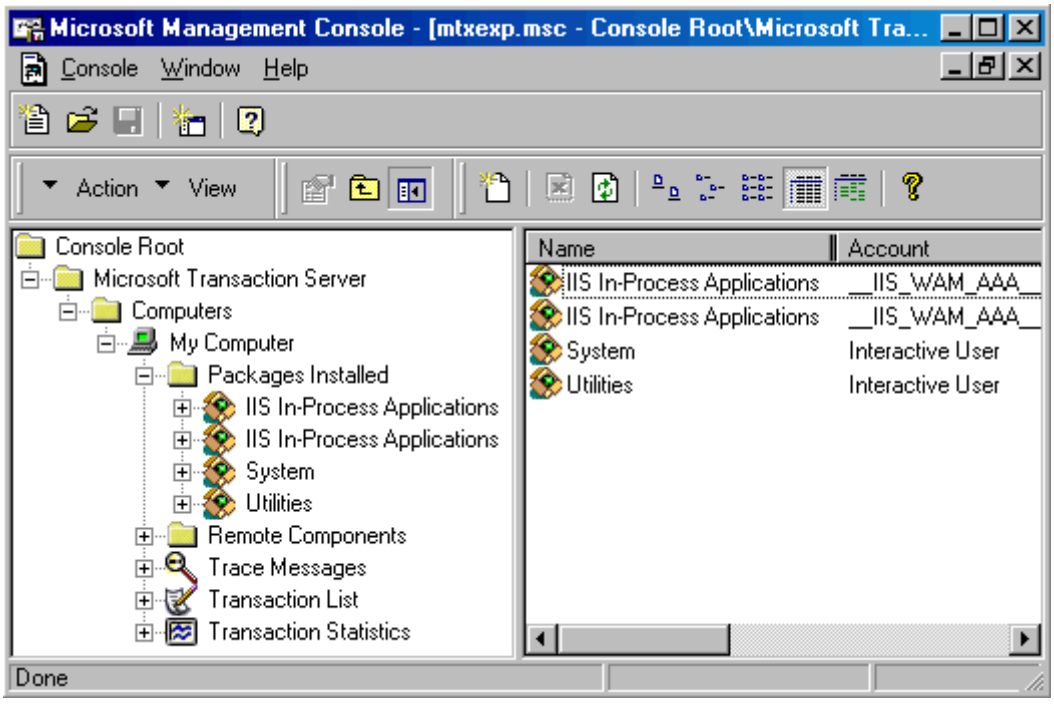
이것으로 간단한 MTS 객체는 완성하였다. 이제 이 프로젝트를 컴파일하면 DLL 파일이 생성될 것이다. 앞 장에서도 설명한 바 있지만, MTS 는 in-process COM 서버를 효과적으로 관리하는 환경이다. DCOM 의 out-of-process 서버를 사용할 때 가질 수 있는 수행 성능의 저하와 리소스 관리의 문제점 등을 효과적으로 관리하도록 해주는 환경으로 MTS 를 이해하면 거의 틀림이 없다.

이렇게 생성된 DLL 을 Run|Register ActiveX Server 메뉴를 이용해 등록하면 개발한 컴퓨터의 in-process COM 서버로 등록할 수 있다. 그렇지만, MTS 환경을 DCOM 을 이용하여 사용하기 위해서는 절대로 이 메뉴를 선택하면 안된다. 그렇게 할 경우 서버 컴퓨터의 in-process DLL 이 실행되는 것이 아니라, 설치한 컴퓨터의 DLL 이 실행되어 버린다.

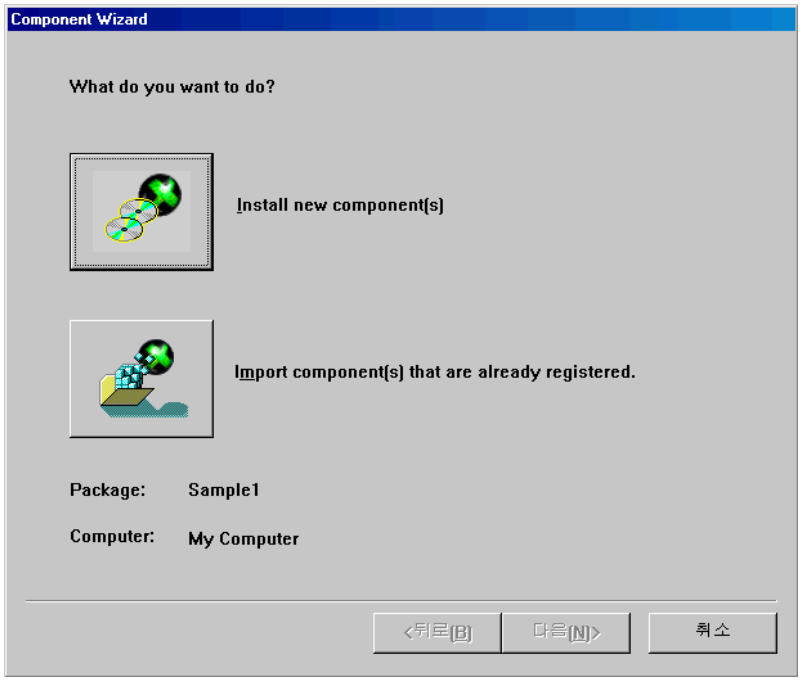
만약, 개발한 컴퓨터가 윈도우 NT 와 MTS 서버를 설치한 환경을 가지고 있으면 Run|Install MTS Objects... 메뉴를 이용하여 MTS 객체를 설치할 수 있지만 필자를 비롯한 대부분의 개발자는 서버에 수시로 접속할 수 있는 클라이언트 개발 환경을 가지고 있을 것으로 사료되므로, 앞 장에서 간단히 설명한 Run|Install MTS Objects... 메뉴를 이용한 객체의 설치방법은 커다란 실효성을 가지지 못할 것이다. 그러므로, 여기에서는 이렇게 만들어진 MTS 객체 DLL 을 서버 컴퓨터에 설치하는 방법을 보다 근본적인 방법을 이용하여 설명하도록 하겠다.

먼저 컴파일을 통해 생성된 DLL 파일을 서버 컴퓨터의 적당한 디렉토리로 복사한다. 필자의 경우 윈도우 NT 서버가 서버 컴퓨터의 D 드라이브에 깔려 있는데, 테스트를 위해 D:₩temp 디렉토리로 MTS 객체 서버 DLL 을 복사하였다. 물론 클라이언트 컴퓨터에 그대로 두고 이 위치에 있는 DLL 서버를 패키지로 등록하여 사용할 수도 있지만, 기본적으로는 여러 컴퓨터가 동시에 사용할 DLL 서버이므로 서버 컴퓨터의 적당한 위치로 옮겨서 사용하는 것이 더 합리적일 것이다. 경우에 따라서는 여러 개의 클라이언트에 MTS 컴포넌트를 분산시켜 위치시킨 후 이를 사용하게 할 수도 있다.

이렇게 적당한 위치에 DLL 파일을 배치했으면, 서버 컴퓨터에서 트랜잭션 서버 탐색기(Transaction Server Explorer)를 실행하도록 한다. MTS 탐색기의 좌측 화면에서 'Package Installed' 폴더를 선택하면 다음과 같이 이미 설치된 패키지들을 볼 수 있을 것이다. MTS 2.0 을 설치한 경우 IIS 에 대한 패키지들도 설치되어 있을 것이다. 참고로 여기서 보여주는 화면은 필자의 서버 컴퓨터에 설치된 윈도우 NT 5.0 베타 1 Korean locale 환경에서 실행된 것이기 때문에, 일반적인 윈도우 NT 4.0 정식 버전에서의 실행 모습과는 다소 다르게 보일 수도 있음을 미리 밝혀둔다.



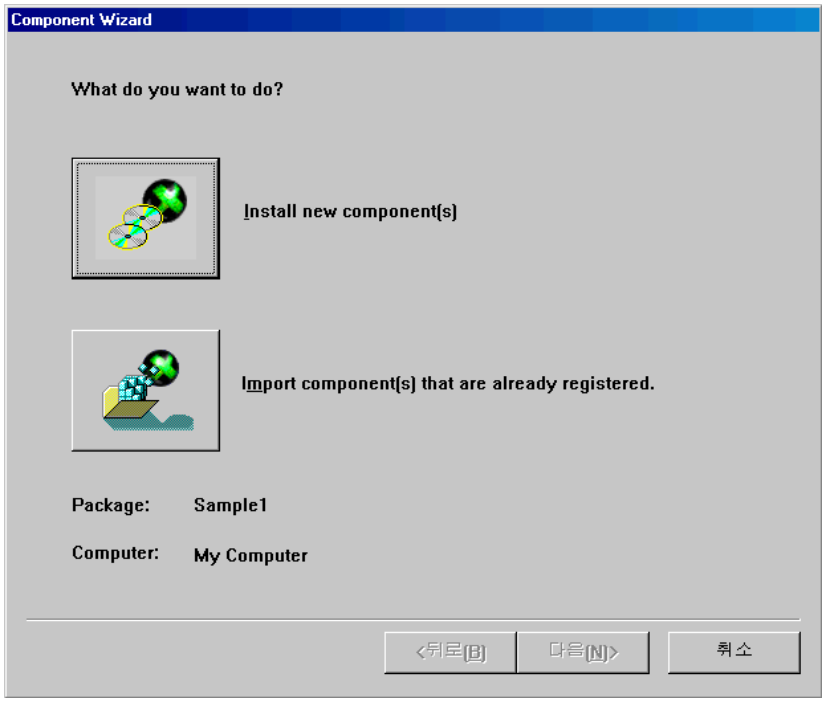
여기에 새롭게 설치할 패키지를 추가해야 한다. 툴바에서 'Create a new object' 버튼을 클릭하거나, 폴더에서 오른쪽 버튼을 클릭하고 New|Package 메뉴를 선택하면 다음과 같이 패키지를 새롭게 작성할 것인지 아니면 작성된 패키지를 설치하는 것인지를 물어온다. 여기에서 우리들은 MTS 객체 컴포넌트를 작성한 것이지 패키지를 작성한 것이 아니므로 'Create an empty package' 버튼을 클릭하도록 한다.



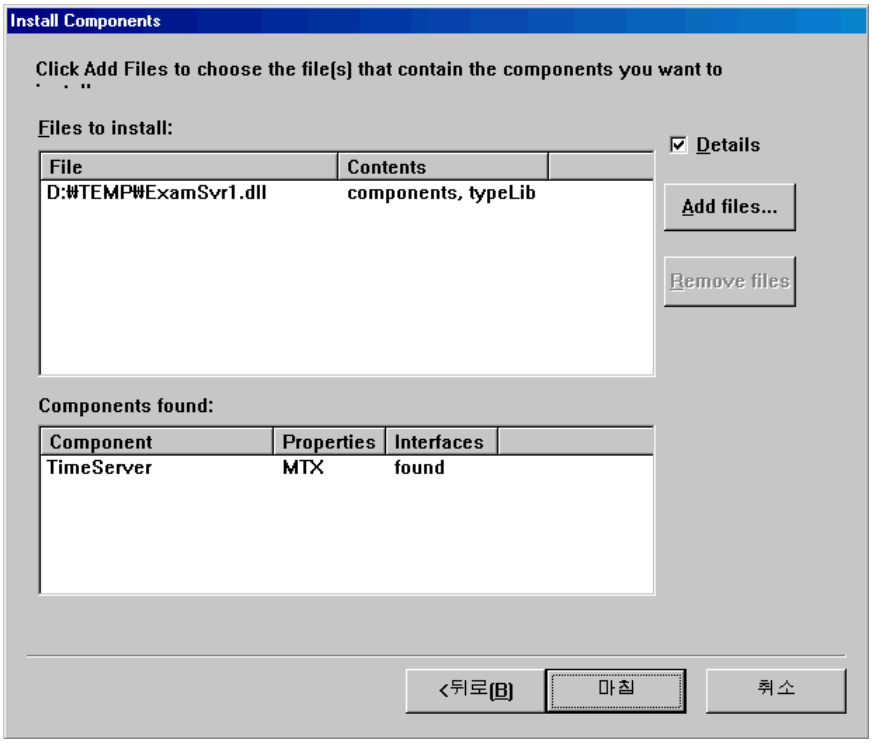
이렇게 하면 새로운 패키지의 이름을 입력하라는 대화상자가 나타나는데, 여기에서 적당한 패키지의 이름을 입력하도록 하자. 필자의 경우는 Sample1 이라는 이름을 부여하였다. 그리고, '다음' 버튼을 클릭하면 이 패키지를 사용하게 될 사용자 계정을 선택할 수 있게 되는데, 여기에서 'Interactive user'를 선택하면 현재 접속한 사용자 들이 접속할 수 있도록 하는 것이며 'This user'를 선택한 경우에는 사용할 사용자를 윈도우 NT 도메인 유저 리스트를 이용하여 선택하고 이들에게 패스워드를 사용하도록 패스워드를 부여할 수 있다. 특수한 보안이 유지되어야 하는 경우를 제외하고는 'Interactive user'를 선택하도록 한다. 그리고, '마침' 버튼을 클릭하면 MTS 탐색기 화면에 Sample1 이라는 새로운 패키지가 설치된 것을 확인할 수 있을 것이다.

이제는 이 패키지에 우리가 작성한 MTS 객체 컴포넌트를 설치할 차례이다. MTS 탐색기의 좌측 pane 에서 Sample1 패키지의 'Components' 폴더를 선택하고 툴바에서 'Create a new object' 버튼을 클릭하거나 오른쪽 버튼을 클릭하고 팝업 메뉴에서 New|Component 메뉴를 선택하여 컴포넌트를 추가한다.

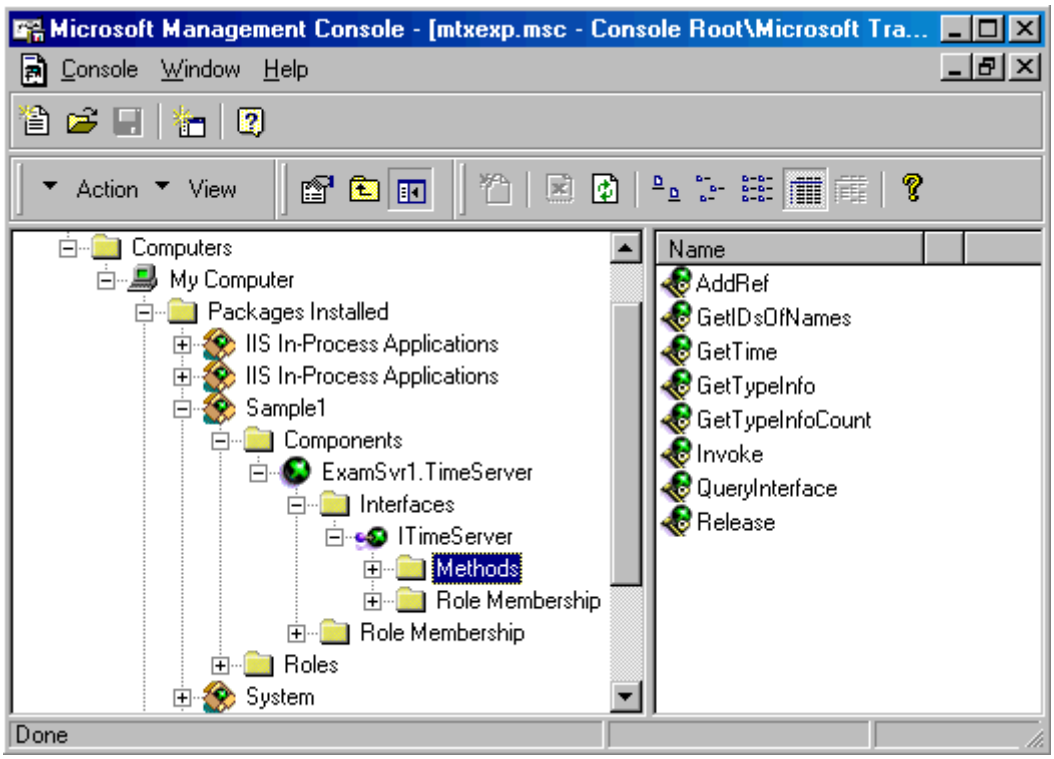
이렇게 하면 다음과 같이 이미 등록된 컴포넌트 중에서 패키지에 등록할 컴포넌트를 고르것인지, 아니면 새로운 컴포넌트를 선택하는 대화상자가 나타날 것이다. 여기서는 새롭게 파일을 복사해서 등록하는 것이므로 'Install new component(s)' 버튼을 클릭한다.



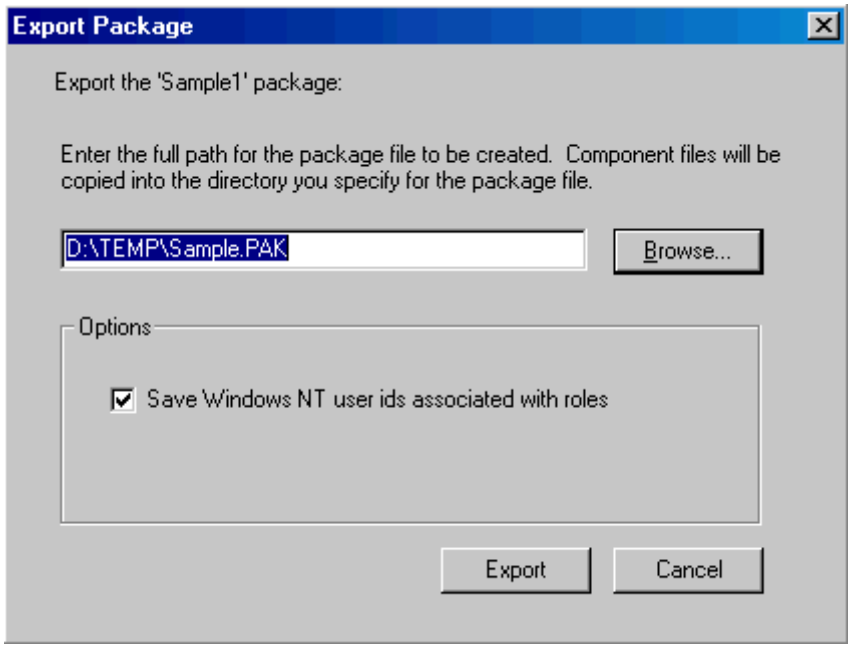
이 버튼을 클릭하면 서버 객체 파일을 추가하는 대화상자가 다음과 같이 나타나게 되는데, 여기에서 우리가 제작한 MTS 컴포넌트 DLL 파일을 'Add files' 버튼을 클릭하여 추가하도록 한다.



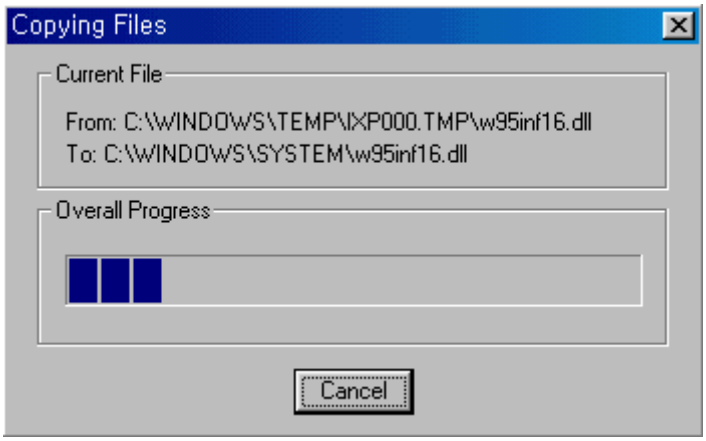
여기서 마침 버튼을 클릭하면 패키지에 우리가 작성한 서버 객체가 설치된다.
다음 화면은 설치된 MTS 컴포넌트를 MTS 탐색기를 이용하여 브라우징한 모습이다.



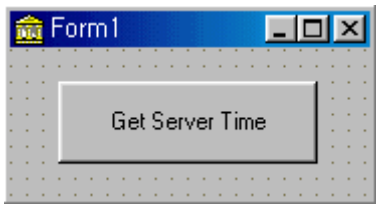
이제 컴포넌트 설치가 끝났으니, 클라이언트에서 사용할 수 있도록 설정해야 한다. 이렇게 서버에 설치한 MTS 패키지를 클라이언트에서 DCOM 을 이용하여 사용할 수 있게 하기 위해서는 클라이언트로 배포할 실행파일을 MTS 서버가 생성하도록 해야 한다. 이를 위해서 배포할 패키지(여기서는 Sample1)를 선택하고 오른쪽 버트을 클릭하면 나타나는 팝업 메뉴에서 Export... 메뉴를 선택한다. 이렇게 하면 다음과 같이 패키지 설치 파일을 생성할 패스와 이름을 입력하라는 대화상자가 나타날 것이다. 여기에 클라이언트 들이 접근할 수 있는 공유된 디렉토리를 패스로 지정하고, 적절한 설치 파일 이름을 다음과 같이 적어 넣는다.



Export 버튼을 클릭하면 해당 디렉토리에 패키지 파일과 함께 패키지 파일의 이름과 동일한 이름을 가진 실행 파일이 담겨 있는 clients 라는 서브 디렉토리가 생성된다.
클라이언트에서 MTS 객체를 실행할 수 있게 하려면 이 clients 서브 디렉토리에 접근하여 이 실행파일(여기서는 sample.exe)을 실행해야 한다. 이렇게 클라이언트 설치 실행파일을 클라이언트에서 실행하면 DCOM 에 의해서 요구되는 프록시-스텁 DLL 들과 타입 라이브러리 파일이 클라이언트 컴퓨터에 복사되고, 클라이언트 시스템의 레지스트리 정보가 여기에 맞도록 업데이트 된다. 이 과정을 통해 클라이언트 어플리케이션이 서버 패키지를 사용할 수 있도록 설저오디는 것이다.
다음 화면은 이 실행파일을 클라이언트에서 실행할 때 나타나는 화면이다. 여기서는 이미 설치한 패키지를 다시 설치하는 화면을 잡았기 때문에 처음 설치하는 화면과는 다소 차이가 있을 것이다.



이것으로 MTS 서버 컴포넌트를 제작하고, 이를 사용할 준비가 모두 끝났다. 이제 이를 실제로 사용하는 클라이언트 어플리케이션을 작성해보자. 델파이에서 New Application 메뉴를 선택하여 새로운 메뉴를 선택하고 폼에 다음과 같이 버튼을 하나 얹도록 하자.

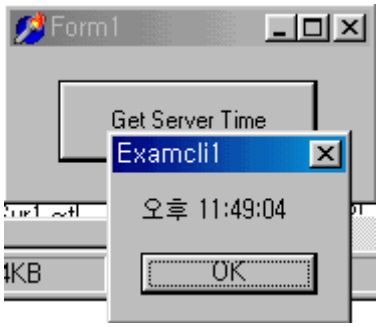


그리고, uses 절에 early 바인딩을 사용할 수 있도록 하기 위해 서버의 타입 라이브러리에 대한 파스칼 유닛(여기서는 ExamSvr1_TLB.pas)을 추가하도록 한다. 물론 late 바인딩을 이용할 경우에는 이를 추가하는 대신 ComObj.pas 유닛을 추가하고 CreateOLEObject 함수를 이용하면 된다.
마지막으로 Button1 의 OnClick 이벤트 핸들러를 다음과 같이 작성하면 클라이언트 어플리케이션은 완성된다.

```
procedure TForm1.Button1Click(Sender: TObject);
var
  TimeServer: ITimeServer;
  Str: WideString;
begin
  TimeServer := CoTimeServer.Create;
  ShowMessage(TimeServer.GetTime);
end;
```

클라이언트 어플리케이션을 실행하고, 버튼을 클릭하여 서버의 시간을 알아보도록 하자.

아마도 첫번째 실행을 할 때 다소의 시간이 걸리겠지만, 이후에는 빠르게 동작하면서 서버의 시간을 알려줄 것이다. 다음은 클라이언트 어플리케이션의 실행 화면이다.



여기서 꼭 짚고 넘어가야 할 부분이 있다.
안타깝게도 필자가 시험해본 결과로는 이렇게 MTS 탐색기를 이용하여 export 한 실행 파일을 실행하여 클라이언트에 MTS 패키지에 대한 정보를 등록하면 윈도우 95 에서는 큰 문제가 없었지만, 윈도우 98 에서는 시스템에 다소 간의 문제가 발생하고 그 다음에 윈도우 98 을 재실행하는 시점부터 DCOM 이 제대로 동작하지 않는 등의 알 수 없는 버그가 존재하였다. 이는 MTS 가 만들어 내는 실행 파일과 윈도우 98 시스템에 호환성이 충분히 고려되지 않았기 때문으로 생각된다.
그러므로, 윈도우 95 를 사용할 때에만 이런 방법을 사용해야 하며, 윈도우 98 을 사용하면 번거럽더라도 제작한 MTS 패키지 컴포넌트에 대한 타입 라이브러리를 직접 클라이언트에서 등록하도록 해야 한다. 아마도, MTS 의 새로운 버전이 나오면 이 문제가 해결될 것으로 생각된다.

MTS 데이터 모듈의 활용

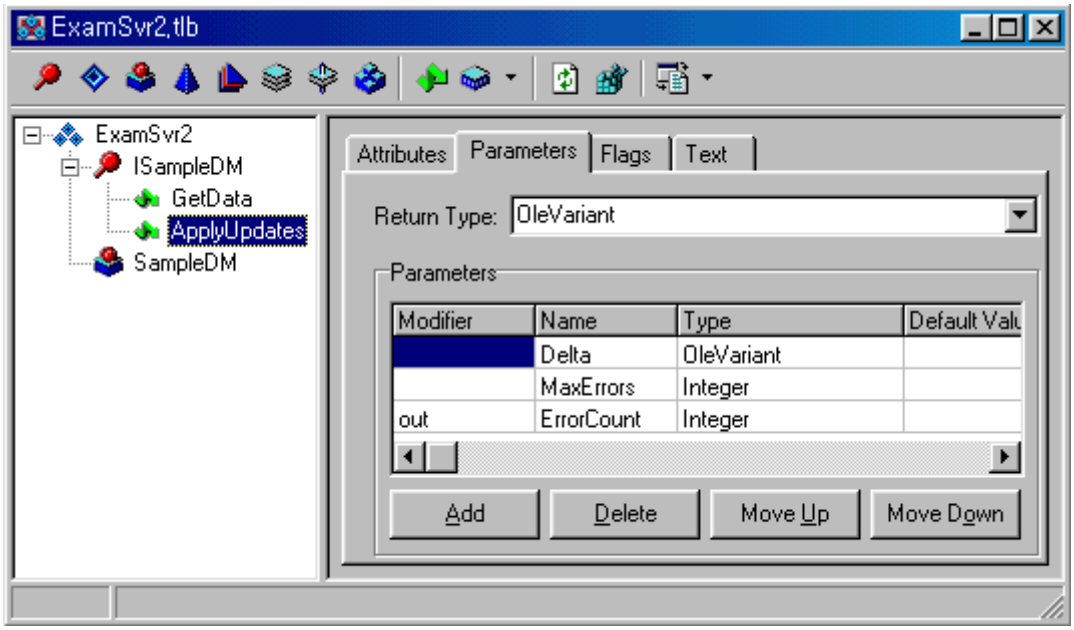
이번에는 데이터 모듈을 이용하여 델파이의 데이터 접근 컴포넌트를 MTS 환경에서 활용하는 방법에 대해서 알아보도록 하자. 기본적인 예제를 작성하기 전에 데이터 모듈을 활용한 데이터베이스 어플리케이션의 작성 요령에 대해서 먼저 간단히 알아보도록 하자.
보통 기본적인 데이터베이스 어플리케이션을 작성할 때에는 TTable, TQuery 등의 데이터 세트 컴포넌트를 폼에 추가하고, 이와 함께 TDataSource 와 각종 데이터 컨트롤을 연결하여 데이터를 데이터베이스에서 불러와서 보여주고, 데이터 컨트롤에서 변경한 내용을 데이터베이스에 접근하는 형태로 어플리케이션을 작성한다.
그렇지만, 보다 확장성이 좋고 견고한 데이터베이스 어플리케이션을 작성할 때에는 OOP 의 캡슐화 개념을 이용하여 모든 데이터베이스에 관련한 작업을 데이터 모듈로 일원화하여 관리하는 것이 좋다. 그것도 단순하게 데이터 모듈을 하나 추가하고, 폼 대신에 데이터 세트 컴포넌트와 데이터 소스 컴포넌트 들을 한 군데 모아 놓고 호출하는 형태로 작업을 하는 것

이 아니라, 데이터 모듈에서 필요로 하는 데이터베이스 작업을 추상화한 뒤에 각각의 데이터베이스 작업을 하나의 메소드로 정의하여 데이터 모듈의 메소드를 호출하면 내부적인 데이터베이스 작업이 이루어지도록 하는 것이 좋다.
이를 달리 해석하면, 데이터 모듈이 데이터베이스 작업을 캡슐화하는 것이다.
MTS 를 활용하여 데이터베이스 작업을 진행하기 위해서는 이러한 개념으로 타입 라이브러리 에디터에서 데이터 모듈에서 수행할 작업들을 인터페이스의 메소드로 추상화하여 정의하고, 이들을 실제로 구현한 뒤에 클라이언트는 인터페이스의 메소드를 호출하여 데이터베이스 작업을 진행하는 구조로 작성해야 한다.
그럼 이런 기본적인 개념에 입각하여 델파이의 DBDEMOS 앨리어스의 biolife.db 데이터베이스 파일의 데이터를 MTS 환경에서 불러오고, 변경된 내용을 업데이트할 수 있는 서버를 작성해보자.
먼저, File|New 메뉴를 선택하고 ActiveX 페이지에서 ActiveX Library 아이템을 더블 클릭하여 새로운 액티브 X in-process DLL 파일 프로젝트를 시작한다. 여기에 File|New 메뉴의 Multitier 페이지에서 MTS Data Module 아이템을 더블 클릭하여 MTS 데이터 모듈 위저드를 시작한다. 이 위저드의 대화 상자는 MTS 자동화 객체 위저드와 완전히 동일하므로 자세한 설명은 생략한다. 여기에서 적당한 데이터 모듈 클래스의 이름을 적어야 하는데, 필자는 SampleDM 이라는 이름을 사용하였다. OK 버튼을 클릭하면 타입 라이브러리 에디터가 나타날 것이다. 여기에서 클라이언트 들이 사용할 메소드를 먼저 정의해야 한다.
이번에 작성할 예제에서는 데이터베이스 파일의 데이터를 클라이언트에서 얻을 때 사용할 GetData 메소드와 클라이언트에서 업데이트한 데이터를 데이터베이스 파일에 적용할 때 사용할 ApplyUpdates 메소드를 정의하도록 한다.
GetData 메소드는 파라미터 없이 호출하면 데이터를 반환해야 하므로, Return Type 으로 OleVariant 를 선택한다. 물론 데이터 모듈에 있는 특정 데이터 필드를 반환할 경우 필드의 데이터 형에 따라 적당한 데이터 형을 선택하면 될 것이다. 이 예제에서 OleVariant 를 선택한 이유는 간단히 TProvider 컴포넌트를 이용하여 데이터베이스 파일의 데이터를 반환할 것이기 때문이다. 그렇지만, 보다 복잡한 데이터베이스 어플리케이션을 작성할 때에는 구체적인 목적을 가진 메소드를 정의하여 해당 데이터 형을 반환하도록 하는 것이 좋다.
예를 들어, 신상정보 데이터베이스에서 특정 ID 를 가진 사람의 이름을 반환하는 메소드를 정의한다고 할 경우에 다음과 같이 메소드를 정의하는 것이 좋을 것이다.

```
GetName(ID: WideString): WideString;
```

이 메소드를 나중에 구현할 때에는 해당 테이블에서 파라미터로 넘어온 ID 값을 이용하여 이름을 알 수 있도록 쿼리를 하거나, 테이블 컴포넌트의 Find 메소드를 이용하면 될 것이다.
ApplyUpdates 메소드는 클라이언트에서 변경한 내용을 데이터베이스 파일에 적용할 때 사용하도록 정의하는데, 여기서는 TProvider 컴포넌트의 ApplyUpdates 메소드를 그대로 사

용할 것이기 때문에, 이 메소드의 파라미터와 반환값을 그대로 사용하면 된다.
이렇게 정의한 타입 라이브러리의 모습은 다음과 같다.



타입 라이브러리 에디터를 닫으면 아마도 다음과 같은 뼈대 코드가 완성되어 있을 것이다. 이 내용은 데이터베이스와 관련된 유닛들이 uses 절에 추가된 것을 제외하면, 기본적으로 MTS 자동화 객체 위저드가 생성하는 것과 동일하므로 자세한 설명은 생략한다.

```
unit U_ExamSvr2;

interface
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  ComServ, ComObj, VCLCom, StdVcl, BdeProv, BdeMts, DataBkr, DBClient,
  MtsRdm, Mtx, ExamSvr2_TLB, Db, DBTables;

type
  TSampleDM = class(TMtsDataModule, ISampleDM)
  private
    { Private declarations }
  public
    { Public declarations }
  protected
```

```
    function ApplyUpdates(Delta: OleVariant; MaxErrors: Integer;
      out ErrorCount: Integer): OleVariant; safecall;
    function GetData: OleVariant; safecall;
  end;

var
  SampleDM: TSampleDM;

implementation

{$R *.DFM}

function TSampleDM.ApplyUpdates(Delta: OleVariant; MaxErrors: Integer;
  out ErrorCount: Integer): OleVariant;
begin
end;

function TSampleDM.GetData: OleVariant;
begin
end;

initialization
  TComponentFactory.Create(ComServer, TSampleDM,
    Class_SampleDM, ciMultiInstance, tmApartment);
end.
```

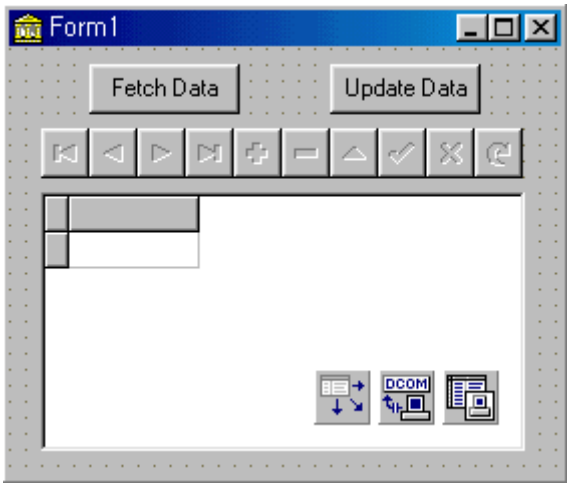
이제 사용할 데이터 접근 컴포넌트 들을 데이터 모듈에 추가하고 이들의 프로퍼티를 설정하도록 하자. 여기서는 간단하게 테이블과 TProvider 컴포넌트로 구현할 것이므로 이들 컴포넌트를 각각 하나씩 데이터 모듈에 추가한다.
그리고, Table1 컴포넌트의 Database 프로퍼티를 'DBDEMOS', TableName 프로퍼티를 'biolife.db'로 설정하고, Provider1 컴포넌트의 DataSet 프로퍼티는 Table1 으로 설정한다.
기본적인 프로퍼티 설정이 끝났으면 다음과 같이 간단하게 인터페이스에서 정의한 2 개의 메소드를 구현한다.

```
function TSampleDM.GetData: OleVariant;
begin
```

```
  Result := Provider1.Data;
  SetComplete;
end;

function TSampleDM.ApplyUpdates(Delta: OleVariant; MaxErrors: Integer;
  out ErrorCount: Integer): OleVariant;
begin
  Result := Provider1.ApplyUpdates(Delta, MaxErrors, ErrorCount);
  SetComplete;
end;
```

주의할 점은 각 메소드의 마지막에 SetComplete 메소드를 호출한다는 점이다. 이 메소드는 MTS 자동화 객체에서도 사용할 수 있는데, 그 의미는 현재 작업한 메소드가 완료되었음을 선언하고, MTS 데이터 모듈의 경우 트랜잭션의 중간에 있다면 이 메소드를 호출함으로써 commit 을 하게 된다. MTS 는 클라이언트가 MTS 데이터 모듈의 메소드를 호출할 때 자동으로 트랜잭션을 시작하기 때문에 이 메소드의 호출이 필요한 것이다.
만약 현재까지의 작업이 잘못 되어서 이를 적용하고 싶지 않은 경우에는 SetAbort 나 DisableCommit 메소드를 이용하면 된다.
이것으로 간단한 MTS 데이터 모듈 객체의 구현이 끝났다. 컴파일을 하면 DLL 파일이 생성되는데, MTS 를 활용하기 위해서는 앞서 설명한 MTS 자동화 객체와 마찬가지로 현재 작성한 컴포넌트를 MTS 패키지에 설치해야 한다.
패키지를 새로 만들어도 되고, 앞서 작성한 패키지에 이 컴포넌트만 새로 추가해도 된다. 여기에 대해서는 이미 앞에서 충분히 설명하였으므로 자세한 설명은 생략하도록 하겠다.
어쨌든 MTS 를 잘 활용하기 위해서는 서버에서 MTS 탐색기를 이용하여 패키지와 컴포넌트를 등록하고 이들의 설정 값들을 잘 관리하는 것이 핵심이라는 것을 다시 한번 강조하고 싶다. 컴포넌트를 패키지에 등록했으면, 클라이언트에서 이들을 사용할 수 있도록 export 하도록 한다. 그리고, 이 과정에서 생성된 clients 서브 디렉토리의 실행 파일을 MTS 클라이언트를 설치하려는 컴퓨터에서 실행하여 기본적인 설정을 마쳐야 클라이언트 어플리케이션을 동작시킬 수 있다.
여기까지의 과정은 이미 상세하게 설명한 바 있으므로, 독자들이 모두 쉽게 설치했을 것으로 믿고 이번에 작성한 서버를 활용하는 클라이언트 어플리케이션의 제작에 들어가도록 하자.
델파이에서 새로운 어플리케이션을 시작하고 다음과 같이 폼에 TDBGrid, TDBNavigator, TClientDataSet, TDataSource 와 TDCOMConnection 컴포넌트를 각각 하나씩 얹고, 버튼을 2 개 추가한다.



그리고, DBNavigator1 과 DBGrid1 의 DataSource 프로퍼티는 DataSource1, DataSource1 의 DataSet 프로퍼티는 ClientDataSet1, ClientDataSet1 의 RemoteServer 프로퍼티는 DCOMConnection1 으로 각각 설정한다.
그리고 가장 중요한 DCOMConnection1 의 ServerName 프로퍼티는 오브젝트 인스펙터에서 드롭 다운 리스트의 형태로 현재 등록된 데이터 모듈 서버를 고를 수 있게 되어 있는데, 여기에서 사용할 서버를 선택한다. 앞서 작성한 서버의 예를 들면 'ExamSvr2.SampleDM' 이 된다. 이렇게 서버를 선택하면 자동으로 ServerGUID 프로퍼티는 해당 서버의 GUID 로 설정된다.
기본적인 설정은 끝났고, 'Fetch Data' 버튼을 클릭하면 원격 서버의 데이터를 DBGrid 에 표시하고, 'Update Data' 버튼을 클릭하면 변경된 데이터를 원격 서버의 데이터베이스 서버에 적용시키도록 코딩하면 된다. 이를 위해서는 서버의 인터페이스 메소드인 GetData 와 ApplyUpdates 메소드를 이용하면 간단하게 구현할 수 있다.
먼저 데이터를 가져오는 Button1 의 OnClick 이벤트 핸들러를 다음과 같이 작성한다.

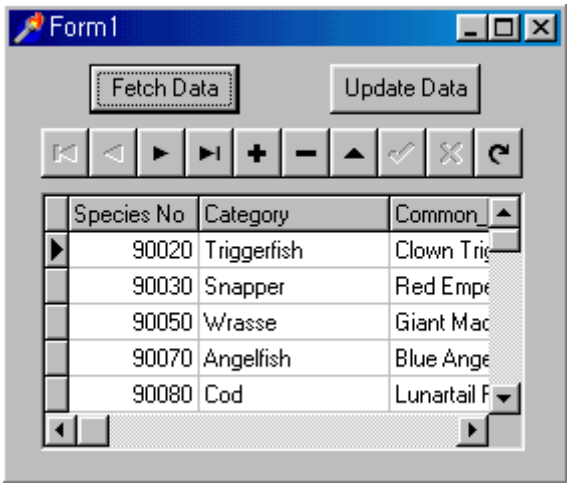
```
procedure TForm1.Button1Click(Sender: TObject);
begin
  DCOMConnection1.Connected := True;
  ClientDataset1.Data := DCOMConnection1.AppServer.GetData;
  DCOMConnection1.Connected := False;
end;
```

즉, 먼저 커넥션 컴포넌트를 원격 서버에 연결하고 AppServer 프로퍼티를 이용하여 서버 컴포넌트의 인터페이스를 얻은 뒤에 이 인터페이스의 GetData 메소드를 호출하면 된다.
이때 클라이언트 데이터 세트의 Data 프로퍼티 역시 OleVariant 데이터 형이기 때문에 쉽게 적용이 가능하다.

마찬가지로 데이터를 업데이트하는 Button2 의 OnClick 이벤트 핸들러는 다음과 같이 작성하여 구현할 수 있다.

```
procedure TForm1.Button2Click(Sender: TObject);
var
  NumErrors: Integer;
  ErrorPackets: OleVariant;
begin
  DCOMConnection1.Connected := True;
  if ClientDataSet1.ChangeCount > 0 then
    ErrorPackets :=
      DCOMConnection1.AppServer.ApplyUpdates(ClientDataSet1.Delta, 2, NumErrors)
  else
    ShowMessage('업데이트할 레코드가 없습니다 !');
  DCOMConnection1.Connected := False;
end;
```

일단 접속을 한 뒤에 변경된 레코드가 있는지 검사하고, 있다면 서버 인터페이스의 ApplyIpdates 메소드를 호출하여 변경된 내용을 전송하게 된다.
간단한 클라이언트 어플리케이션의 작성이 완료되었다. 이제 이를 컴파일하고 실행하면 다음과 같이 원격 서버와 함께 동작하는 데이터베이스 클라이언트 어플리케이션의 화면을 볼 수 있을 것이다.



정 리 (Summary)

이번 장에서는 구체적인 MTS 패키지와 컴포넌트를 서버와 클라이언트에 설치하고, 이를 활용하는 방법과 델파이 4 에서 제공하는 위저드를 이용하여 실제로 MTS 컴포넌트를 작성하는 방법에 대해서 알아보았다.

필자의 개인적인 생각으로는 MTS 환경은 CORBA 환경에 비해 가벼우면서도 강력한 기능을 제공하는 우수한 환경으로 볼 수 있을 것 같다. 엔터프라이즈 환경이 아닌 적당한 규모에서, 특히 중소 규모의 네트워크 환경에서 MTS 를 이용하여 분산 환경을 구축한다면 가장 저렴하면서도 적합한 솔루션이 될 것으로 믿는다.

델파이 4 에서 제공하는 기능 중에서도 가장 실용적이고, 우수한 위저드인 MTS 관련 위저드를 적절하게 활용하는 것이 바로 분산환경에 쉽게 적응하는 지름길이 될 것이므로, 여기에 대한 공부를 소홀히 하지 않기를 바라는 바이다.