
실전 DataSnap !

작성자 : Bob Swart

2011 년 3 월 번역 (원문: 2011 년 1 월)

Americas Headquarters
100 California Street, 12th
Floor
San Francisco, California
94111

EMEA Headquarters
York House
18 York Road
Maidenhead, Berkshire
SL6 1SF, United Kingdom

Devgear
서울특별시 반포 1 동 743-14
4 층 (주)데브기어
(T) 02.595. 4288

실전 DATASNAP !

지난해, 델파이 2010 엔터프라이즈와 그 이상에서 지원되는 DataSnap에 대한 기술 백서를 발표하였습니다 (이 기술 문서의 마지막 부분에서 링크를 확인하실 수 있습니다). 그 기술 백서에서는 DataSnap 와 멀티-티어 개발에 도움이 되는 기초 지식을 소개하였습니다. 그러나 몇 가지 중요한 개선 사항과 향상된 점이 있습니다. 기존의 기술 백서를 다시 쓴다기 보다는 DataSnap을 기존의 설명이 아닌 새로운 점과 향상된 기능들을 순서에 따라 특별한 방법으로 어떻게 사용하는지를 기술하는 것이 도움이 될 것입니다

이 기술 백서에서 다루게 될, 아래와 같은 특징들을 포함한 새로운 DataSnap XE 기능들입니다:

- DataSnap 서버 위저드들
- ISAPI DLLs를 위한 HTTPS 지원
- 암호화 필터 (HTTPS를 사용하지 않을 때)
- 향상된 TAuthenticationManager 컴포넌트
- 서버 메소드를 위한 룰에 기초한 인증
- TDataSetProvider를 위한 룰에 기초한 인증
- TDSSessionManager
- (IIS상에서) HTTPS/SSL를 사용한 개발

또한 DataSnap의 이전 버전에서 이미 있었으나 기술백서에 소개되지 않았던 기능과 특징에 대해서도 다룰 것입니다. (비록 그 내용들은 제가 작성한 DataSnap 개발 코스 매뉴얼에 자세하게 언급되고 있지만 – 이 문서의 마지막에 링크 연결을 확인하실 수 있습니다). 주제는 다음과 같습니다:

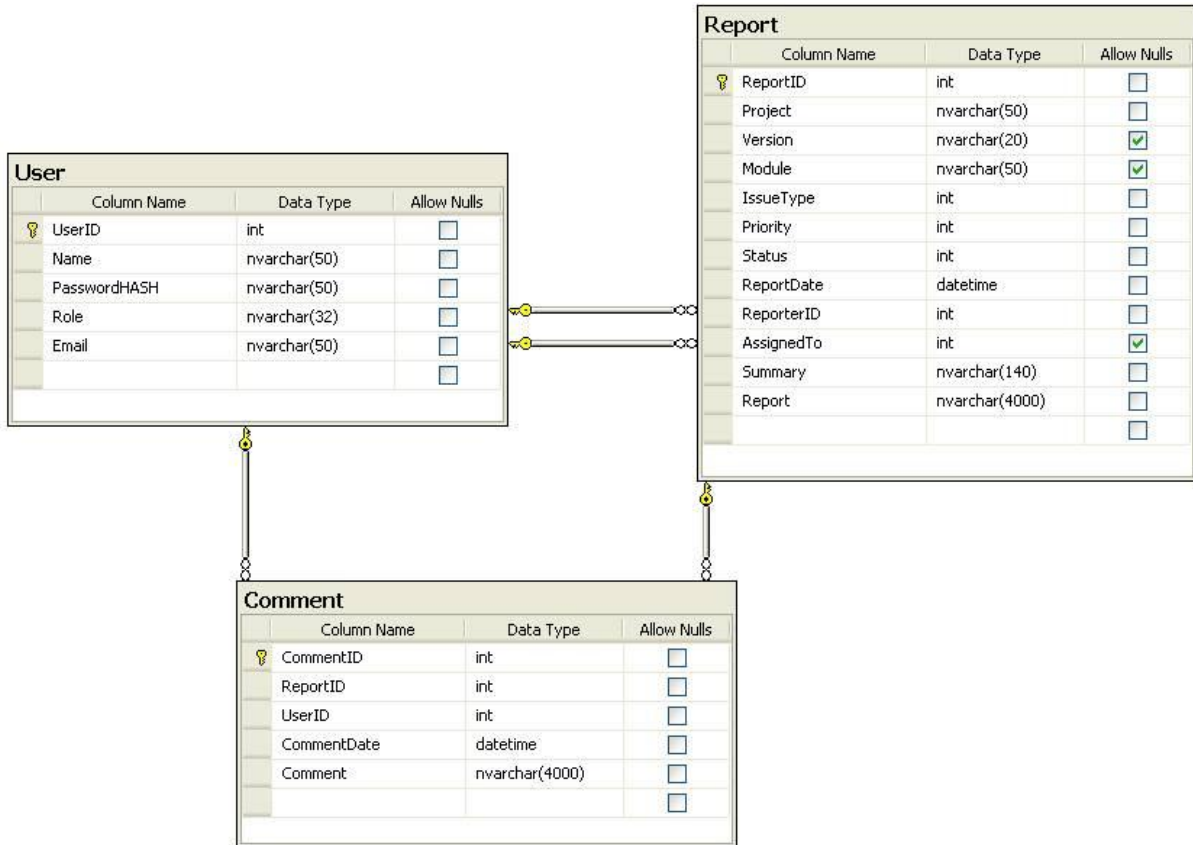
- 노출된 서버 모듈 메소드를 숨기는 방법
- 자동 증가 키 필드
- DBX 와 DataSnap를 이용하여 마스터-상세 연결
- 클라이언트에서 서버로 파라미터 값 전달
- 서버 측에 파라미터 값 지정

이 문서에서는, 규모는 작지만 짧은 시간에 개발하여 실제 사용할 수 있는 현실적인 DataSnap 어플리케이션에 대해 설명하겠습니다.

이 어플리케이션은 개발 이슈 보소서 틀을 대표하는 D.I.R.T.라 불리는, 시스템을 모니터링하고 추적하는 업무입니다. 이러한 시스템은 하나의 프로젝트에 이슈 보고서를 입력하기 위해서 이미 개발자들에 의해 사용되며 (다른 상황에서), 개발자들은 문제가 보고되고 수정이 필요한 경우 “먼지까지도 지워야 한다”고 종종 농담을 합니다

데이터 모델

어플리케이션을 작성하기 전에, 이러한 개발자들이 이슈들을 리포트하고 작업할 수 있고 기술할 수 있는 데이터 모델을 설계해야 합니다. 데이터 모델의 설계 과정은 사용자와 함께 하는 보통 브레인스토밍 회의가 포함됩니다 이러한 경우 다음과 같은 결과를 냅니다:



유저(사용자)

사용자 테이블은 테스터와 어플리케이션에 접근해야 하는 다른 사용자 들 뿐 만 아니라 개발자들의 리스트를 포함합니다. 반드시 입력되어야 하는 정보는 유저 이름과 (해쉬 된) 패스워드, 이 메일 주소입니다. 주소 또는 전화번호와 같은 다른 필드들은 필요한 경우에 추가하면 됩니다. 예제에서 유저는 “개발자”, “감수자” 또는 “관리자” 와 같은 롤을 가질 수 있습니다. 시스템은 특별한 “admin” 롤을 추가하여 이러한 롤 이름들을 사용합니다. 필요 시 유저에 대한 복수 개의 롤을 분리하기 위해 콤마를 사용하여 확장할 수 있습니다.

```
CREATE TABLE [dbo] . [user] (
    [UserID] [int] IDENTITY (1, 1) NOT NULL,
    [Name] [nvarchar] (50) NOT NULL,
    [PasswordHASH] [nvarchar] (50) NOT NULL,
    [Role] [nvarchar] (32) NOT NULL,
    [Email] [nvarchar] (50) NOT NULL,
)
```

유저의 실제 패스워드는 저장할 필요가 없지만 패스워드의 해쉬 값은 있어야 합니다. 이 것은 패스워드의 해쉬 된 버전이 단지 클라이언트에서 어플리케이션 서버 부분으로 연결부분에 넘어간다는 것을 의미합니다. 간단히 해쉬가 어떻게 생성되는지 어떻게 컨넥션이 안전하게 보장되는지 보실 수 있습니다.

보고서

보고서 테이블은 가장 상세한 데이터 셋입니다. 여기서, 새로운 보고서를 입력할 수 있어야 하고 (104문자의 간략한 요약 – 뿐만 아니라 트위터 되도록), 우선 순위뿐만 아니라 프로젝트, 버전(옵션), 모듈(옵션), 이슈 타입 등을 기술할 수 있어야 합니다. 특별한 필드는 이슈의 상태를 포함하는 (보고됨, 할당됨, 오픈됨, 해결됨, 테스트됨, 배포됨, 종료됨과 같이). 또 중요한 것은 리포트의 날짜 인데, 비록 후에 선택 사항이지만 리포트 되는 유저, 이슈가 할당되는 유저, 문제를 해결해야 하는 유저가 시작부터 바로 알려지지 않을 수 있기 때문입니다

```
CREATE TABLE [dbo] . [Report] (
    [ReportID] [int] IDENTITY (1, 1) NOT NULL,
    [Project] [nvarchar] (50) NOT NULL,
    [Version] [nvarchar] (20) NULL,
    [Module] [nvarchar] (50) NULL,
    [IssueType] [int] NOT NULL,
    [Priority] [int] NOT NULL,
    [Status] [int] NOT NULL,
    [ReportDate] [datetime] NOT NULL,
    [ReporterID] [int] NOT NULL,
    [AssignedTo] [int] NULL,
    [Summary] [nvarchar] (140) NOT NULL,
    [Report] [nvarchar] (4000) NOT NULL
)
```

ReportedID 와 AssignedTo 필드들은 데이터 모듈의 다이어그램에서 그려진 것처럼 실제 유저 테이블의 외래 키이며 이슈 타입, 우선순위와 상태 필드들은 정수형임을 주목하십시오. 예를 들어 1부터 10까지의 범위를 사용할 수 있는 이러한 값들의 실제 의미는 다른 곳에서 정의되어야 합니다. 문자열 의미를 저장하기 위해 추가적인 참조(Look-Up) 테이블들을 사용했을 수도 있지만 표시하는 우리 시스템에서는 단지 정수 값만 사용합니다.

한 가지 규칙에 동의해야 합니다: IssueType 와 Priority 는 1부터 10까지의 범위이며 1 이 10보다 적은 수이기 때문에 Priority 1의 Type 1 인 이슈보다 Priority 10의 Type 10인 이슈가 더 긴급합니다

Status용으로 이슈가 가질 수 있는 상태 값들의 초기 리스트를 정의하였습니다:

상태	의미
1	리포트됨
2	할당됨
3	오픈됨
6	해결됨
7	테스트됨
8	배포됨
10	닫힘

“실행중”, “더 많은 정보가 필요함”, 과 “확인됨”과 같은 상태 (배포 후에)등을 가지고 시스템을 확장하기 위해서 현재와는 약간의 의도적인 차이가 있다는 것에 주목하십시오 이런 상태는 지금

당장은 필요하지 않지만 나중에 추가하고자 할 수도 있습니다.

상태 값의 변환과 문자열 표현을 위한 참조 테이블을 추가 할 수 있지만, 화면 GUI 에서 처리 하도록 하겠습니다.

코멘트(주석)

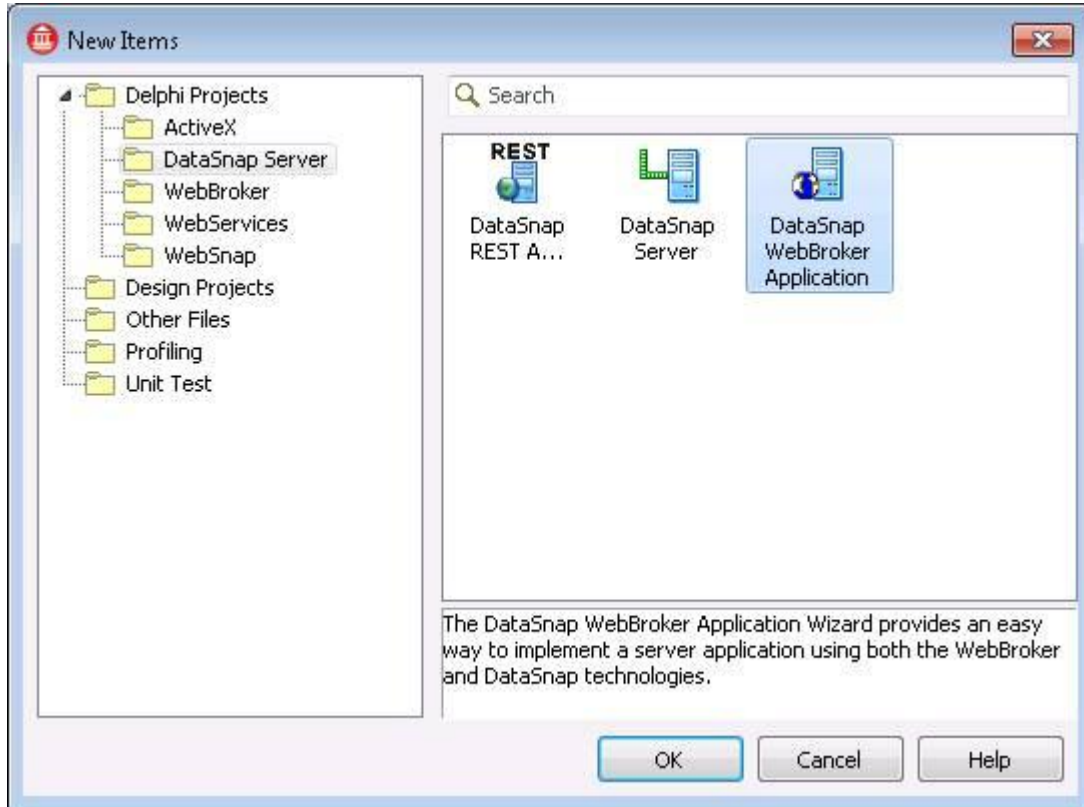
코멘트 테이블은 문제에 대해 무엇이 보고되고 처리되었는지 보고하는 실제로 워크플로우 또는 로그일지를 포함합니다. 이 것은 각 문제 보고서에 대해, 사용자가 주석을 (실제 진행 상황을 보고 특정 날짜에)달 수 있다는 것을 의미합니다.

```
CREATE TABLE [dbo] . [Comment] (
    [CommentID] [int] IDENTITY (1, 1) NOT NULL,
    [ReportID] [int] NOT NULL,
    [UserID] [int] NOT NULL,
    [CommentDate] [datetime] NOT NULL,
    [Comment] [nvarchar] (4000) NOT NULL,
)
```

다른 ReportID, ReporterID, AssignedTo 와 UserID 필드들에 의해 서로 연결되는 이러한 세 개의 테이블들은 D.I.R.T. 프로젝트용 DataSnap 서버 어플리케이션을 작성하기 위해 필요한 전부입니다. 명백히, 더 많은 테이블들을 추가 되어 질 수 있는데 그 목적은 가능한 한 간단하게 최적의 기능을 얻기 위함입니다.

DATA SNAP 서버

델파이 XE 엔터프라이즈 (혹은 아키텍트)는 DataSnap 서버 생성을 도와주는 오브젝트 리포지토리에서 최소 세 개의 새로운 위저드를 제공합니다.

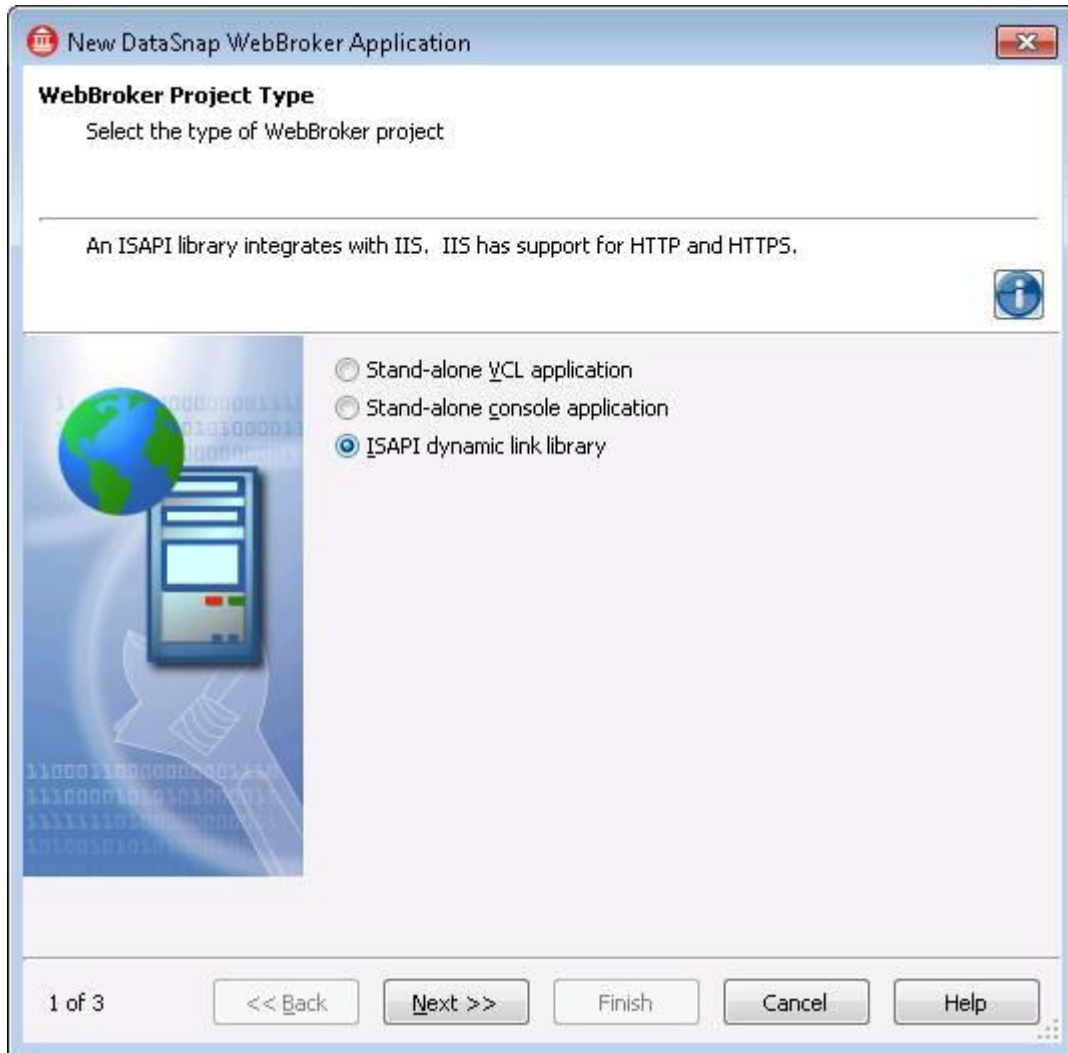


D.I.R.T. 어플리케이션을 위해, 안전한 방법으로 전 세계 각국에서 액세스 할 수 있는 DataSnap 서버 타입이 필요합니다(문제 보고서 중 일부는 우리가 모든 사람에게 노출하고 싶지 않은 내부 세부 사항이 포함 될 수 있습니다)

서로 다른 모바일 기기와 무선 연결에서 액세스를 허용하려면, 암호화 통신 및 서버의 보안 확인을 제공하기 위해 SSL / TLS 를 인증서를 사용하여 HTTPS 를 (HTTP 를 보안)의 사용 프로토콜이 선정되었습니다.

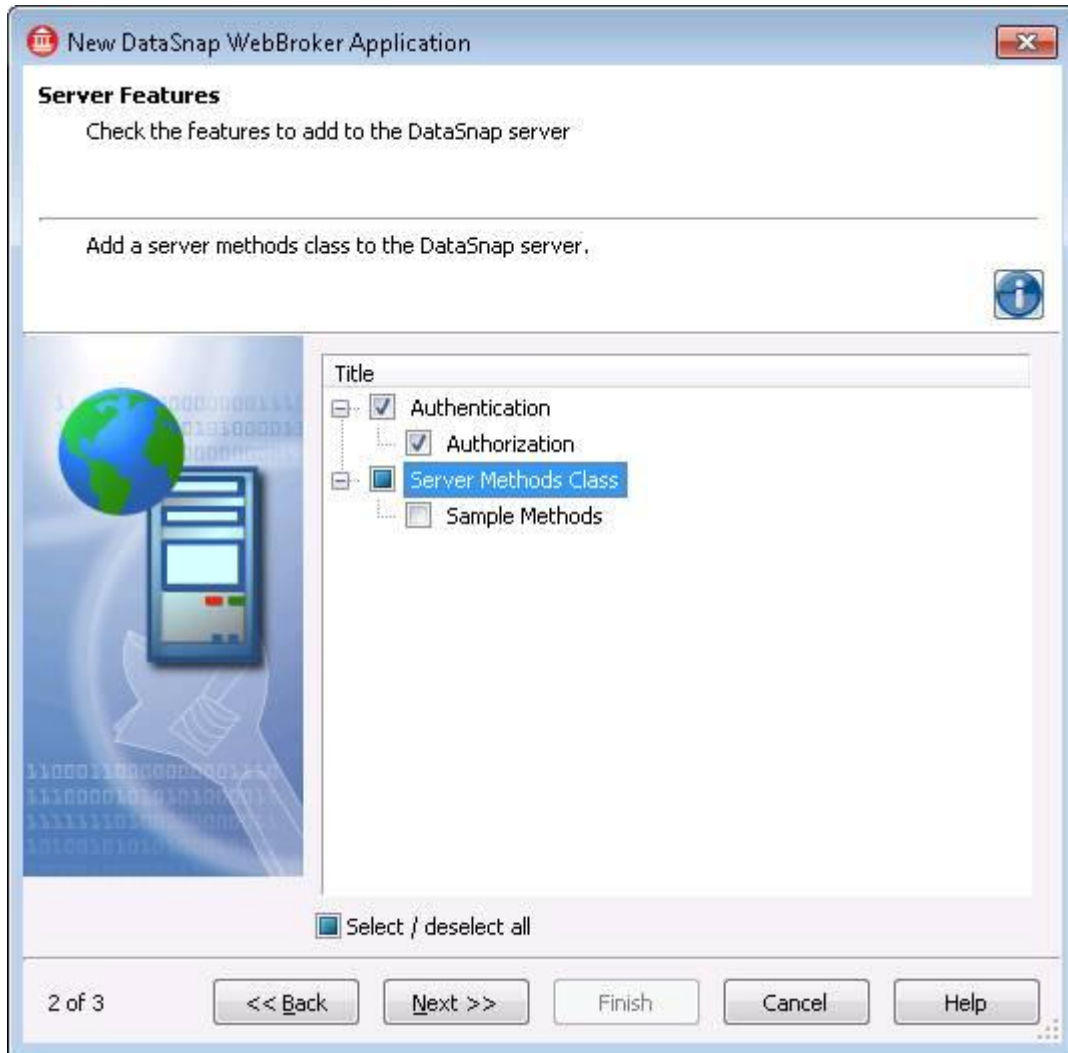
HTTPS를 선택한 이유는 HTTPS는 HTTP보다 더 안전하며 TCP/IP가 HTTP 나 HTTPS보다 빠르지만 속도는 이 어플리케이션에서 문제가 되지 않기 때문입니다. 만일 더 빠른 속도가 필요하다면, TCP/IP 통신 프로토콜을 선택하는 것이 더 좋을 것 입니다.

HTTPS를 선택한다는 것은 두 개의 DataSnap 서버 프로젝트 대상을 의미합니다: DataSnap 웹 브로커 또는 DataSnap REST 어플리케이션 위저드에 의해 생성되는 ISAPI DLL입니다. 두 경우의 차이점은 REST 어플리케이션을 위한 추가적인 파일 (자바 스크립트, 템플릿, 스타일 시트와 이미지)들의 작성입니다. 이러한 추가적인 파일들은 지금은 필요하지 않고, DataSnap 웹 브로커 어플리케이션 위저드를 사용하여, ISAPI 동적 연결 라이브러리(DLL)을 생성합니다. 이 ISAPI DLL은 이 문서의 뒤 부분에서 언급하겠지만 마이크로소프트의 인터넷 정보 서비스 (IIS)상에 배포될 것입니다.

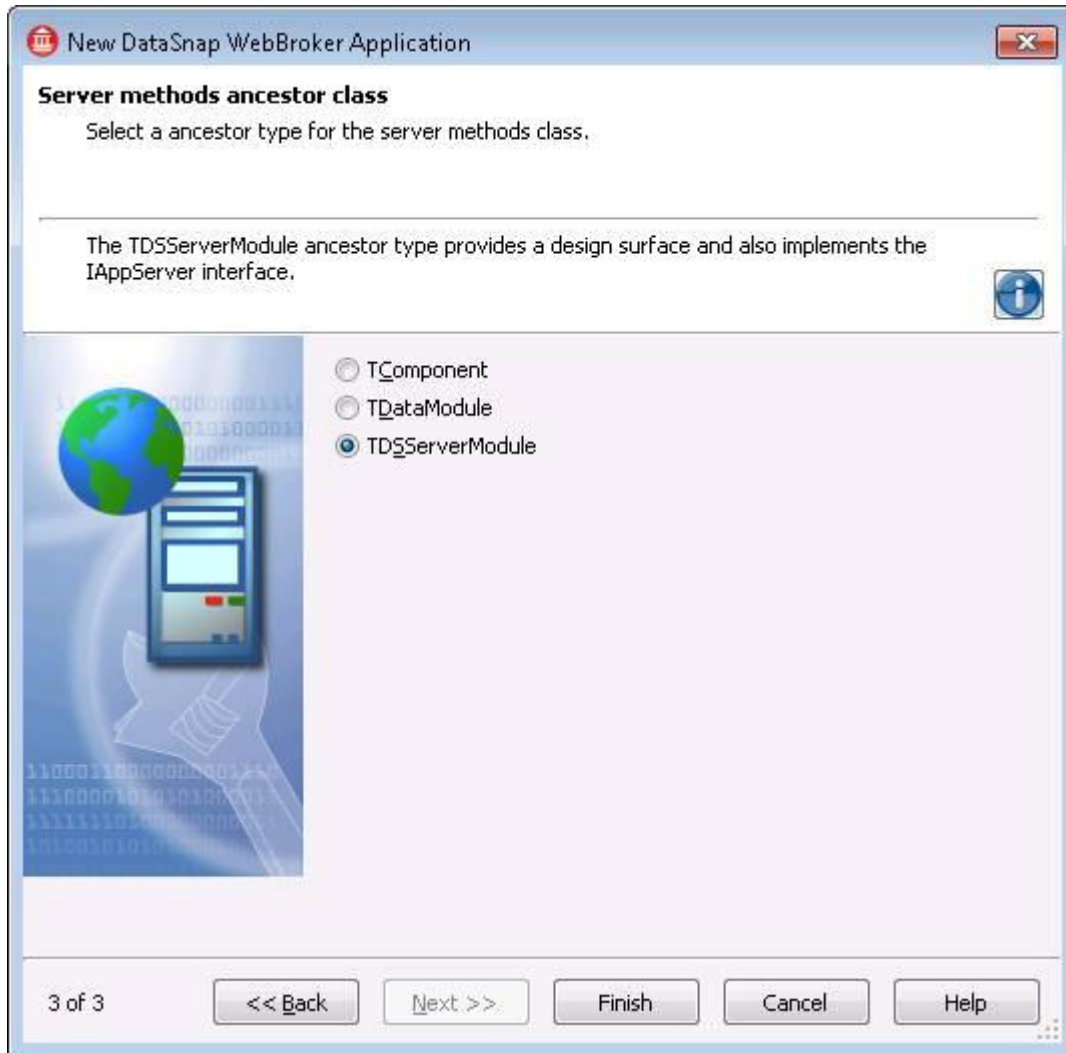


DataSnap 웹 브로커 어플리케이션의 두 번째 페이지에서, 인증과 권한부여를 사용하고자 하거나 샘플 메소드들을 선택할 수 있는 서버 메소드 클래스가 필요하다면 지정할 수 있습니다. 권한 부여가 기능이 유저 롤에 의해 명시적으로 허용되는지 금지되는지를 컨트롤하는 반면 인증은 (유저명과 패스워드를 조합하여) 사용자가 있는 경우 사용자가 누구인지 체크하는 것을 의미합니다.

이 경우는, 서버 메소드뿐만 아니라 인증과 권한이 모두 필요하지만, 필요한 자체 서버 메소드를 쉽게 추가 할 수 있기 때문에 샘플 메소드는 필요하지 않습니다:



DataSnap 웹 브로커 어플리케이션 위저드의 마지막 단계에서, 서버 메소드 클래스의 상위 클래스를 지정할 수 있습니다. 서버 메소드와 데이터셋을 익스포트 하기 위해서 (TDataSetProviders를 사용하여), TDSServerModule를 선택해야 합니다. TComponent 상위 클래스를 사용하면, 서버 메소드를 정의 할 수 있으며, TDataModule를 사용하면 비-시각적인 컴포넌트들을 추가할 수 있으나 TDSServerModule는 시각적으로 보이지 않는 TDSProviderDataModuleAdapter 클래스를 사용하여 IAppServerFastClass 인터페이스 메소드들을 구현하는 선조 클래스입니다.



Finish 버튼을 클릭하면, ServerMethods 유닛과 웹 모듈의 유닛을 포함한 두 개의 소스 파일을 갖는 새로운 프로젝트가 생성됩니다. 프로젝트를 DirtServer로 저장합니다 (혹은 D.I.R.T. 프로젝트에 원하는 어떤 이름으로) 서버 메소드 유닛은 DirtServerMethods.pas로 웹 모듈은 DirtWebMod.pas로 저장합니다.

서버 메소드 유닛의 이름을 변경하여서 프로젝트가 컴파일되지 않을 것임에 유의하십시오. 웹 모듈 유닛의 uses 절의 ServerMethodsUnit1를 DirtServerMethods로 변경해야만 합니다.

같은 웹 모듈의 OnGetClass 이벤트 핸들러에서 PersistentClass를 지정하는 곳에도 유닛 이름을 적용하셔야 합니다.

```

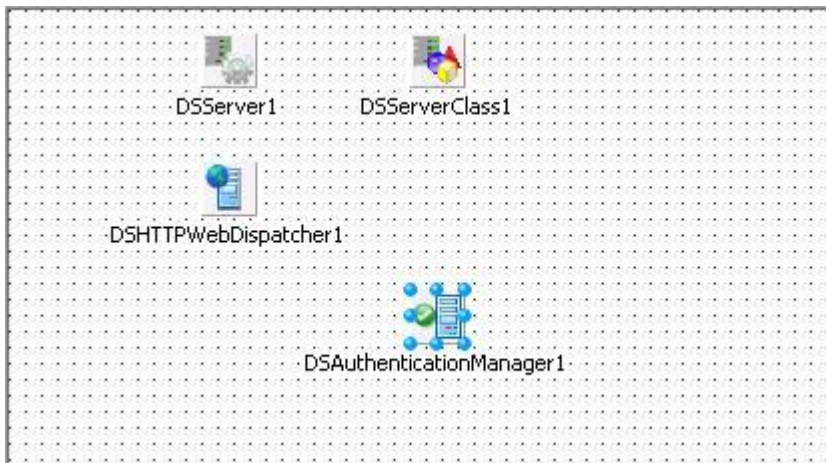
procedure TWebModule1.DSServerClass1GetClass (
  DSServerClass : TDSServerClass;
  var PersistentClass : TPersistentClass);
begin
  PersistentClass := DirtServerMethods.TServerMethods1;
end;

```

이러한 두 가지 수정을 하면 DirtServer DataSnap 서버 프로젝트가 컴파일 되고 ISAPI DLL을 생성합니다. 이제부터 인증 체크를 시작으로 추가된 실제 기능들에 대해 설명하겠습니다.

로그인과 인증

앞에서 수정한 웹 모듈은 DataSnap 서버의 인증과 인증을 컨트롤 할 수 있는 곳입니다. 새로운 TDSAuthenticationManager 컴포넌트는 이 작업의 핵심입니다.



TDSAuthenticationManager 컴포넌트는 사용된 프로토콜뿐만 아니라 (HTTP가 아닌 확실히 HTTPS인) 유저, 패스워드가 유효한지 검증할 수 있는 (DIRT 데이터베이스에서 유저 테이블에 대해서) OnUserAuthenticate 이벤트 핸들러를 가지고 있습니다. 통신 프로토콜로 보안 HTTPS사용하는지 확인하기 위해 초기 구현은 아래와 같이 시작할 수 있습니다.

```

procedure TWebModule1.DSAAuthenticationManager1UserAuthenticate (
  Sender : TObject; const Protocol, Context, User, Password : string;
  var valid: Boolean; UserRoles: TStrings ) ;
begin
  valid := LowerCase (Protocol) = 'https' ;
  // now also validate User and Hashed password...
end;

```

다음 단계는 유저 테이블에 이 조합이 존재하는 적절한 레코드를 확인하는 유저와 (실제로 패스워드의 해쉬 된 값을 포함해야 하는) 패스워드를 포함합니다 이 경우에, 유저 테이블상에서 직접 실행을 수행하기 위한 목적으로만 로컬 TSQLConnection 컴포넌트를 사용할 수 있습니다. 교대로, 유저명과 해쉬 된 패스워드를 서버 머신 상의 설정 파일 또는 ini 파일에 저장할 수 있고. 유저가 로그인 하고자 할 때 그 파일을 읽습니다.

데이터베이스에 직접 연결한다고 가정해보자 TSQLConnection 컴포넌트를 웹 모듈에 위치시키고 Driver 속성을 MSSQL로 지정하고, 데이터베이스와의 연결을 검증하기 위해 Hostname, Database, UserName 와 Password속성을 지정합니다. LoginPrompt 속성을 False로 지정하고, 성공적으로 연결되는지 보기 위해서 Connected 속성을 False에서 true로 변경합니다. TSQLConnection과 별도로 유저 테이블에서 유저와 패스워드를 검색하기 위한 쿼리를 실행하기 위해 TSQLDataSet가 필요합니다.

로컬 DB 연결

웹 모듈에 TSQLConnection 과 TSQLDataSet 컴포넌트를 올려 놓지 않는다면 동적으로 생성할 수 있습니다. 특히 TSQLConnection 가 로그 인을 확인하기 위해 한 번만 필요한 경우라면, (독자들의 연습으로 남겨놓은) 특별히 SQLConnection 매개변수 설정이 .ini 나 config 파일에서 읽을 수 있도록 고려한다면 이 것은 완벽한 해결책입니다.

두 가지 경우 모두, 유저 테이블에서 UserID와 Role의 SELECT을 수행하기 위해 필요한 SQL 쿼리는 WHERE 절에서 유저와 (해쉬 된) 패스워드를 넘깁니다. 이미 해쉬 된 포맷으로 보내졌기 때문에 (클라이언트에서 해야 할) 여기에서 해쉬 패스워드를 가질 필요가 없습니다.

만일 SELECT 문이 레코드를 리턴 한다면, 현재 세션에 UserID를 저장 할 수 있고, 현재 쓰레드의 세션을 구하기 위해 TDSSessionManager.GetThreadSession를 사용하며, "UserID"라는 이름의 필드에 UserID값을 써 넣기 위해 PutData를 호출합니다. UserRoles 컬렉션에 추가하는데 사용할 수 있는 연관된 룰이 SELECT 문에 의해 검색됩니다.

```

procedure TWebModule1.DSAutehnticationManager1UsreAuthenticate (
  Sender: TObject; const Protocol, Context, User, Password: string;
  var valid: Boolean; UserRoles: TStrings) :
var
  SQLConnection: TSQLConnection;
  SQLQuery: TSQLQuery;
  Role: String;
begin
  valid := LowerCase (Protocol) = 'https';
  if valid then
    begin // validate User and Hashed password
      SQLConnection := TSQLConnection.Create (nil);
      try
        SQLConnection.LoginPrompt := False;
        SQLConnection.DriverName := 'MSSQL';
        SQLConnection.Params.Clear;

        // The following parameters can be read from a config file
        SQLConnection.Params.Add('HostName=.');
        SQLConnection.Params.Add('Database=DIRT');
        SQLConnection.Params.Add('User_name=sa');
        SQLConnection.Params.Add('Password=*****');

        SQLQuery := TSQLQuery.Create(nil);
        SQLQuery.SQLConnection := SQLConnection;
        SQLQuery.CommandText :=
          'SELECT UserID, Role FROM [User] WHERE ' +
          ' (Name = :UserName) AND (PasswordHASH = :Password)';

        SQLQuery.ParamByName('UserName').AsString := User;
        SQLQuery.ParamByName('Password').AsString := Password;
        try
          SQLQuery.Open;
          if not SQLQuery.Eof then
            begin
              valid := True;
              TDSSessionManager.GetThreadSession.PutData('UserID',
                SQLQuery.Fields[0].AsString);
              Role := SQLQuery.Fields[1].AsString;
              if Role <> '' then
                UserRoles.Add(Role)
            end
          end
    end

```

```

    else
        valid := False
    finally
        SQLQuery.Close;
        SQLQuery.Free
    end
finally
    SQLConnection.Connected := False;
    SQLConnection.Free
end
end
end;

```

이 구현은 유저당 하나의 롤 만을 허용한다는 것에 주목하십시오. 데이터모델(UserRoles 연결뿐만 아니라 롤을 정의하는)에 더 많은 테이블을 추가하거나 롤 필드에 콤마를 사용하여 롤을 분리하여 확장 할 수 있습니다. 두 가지 확장은 독자들의 연습용으로 남겨놓겠습니다(DataSnap XE 개발자 기초 교육용 매뉴얼에서 다루게 될 것입니다).

권한부여(AUTHORIZATION)

인증이 되었기 때문에, 권한 부여 또한 처리되어야 합니다. DataSnap 서버 클래스와 서버 메소드를 유저 롤로 연결할 수 있는 기능을 제공합니다. DSAAuthenticationManager 컴포넌트의 OnUserAuthorize 이벤트는 권한 부여를 처리하기 위한 구현방법으로 사용됩니다. 기본적으로, 권한을 부여하는 방법은 두 가지가 있습니다: 낙관적이거나 비관적인 접근방법이 있는데 낙관적이라 함은 명백하게 금지하지 않은 어떠한 오퍼레이션이 다 허용된다는 것입니다. 반면 비관적인 접근은 명백하게 허용하지 않는 한 모든 오퍼레이션이 허용되지 않습니다. 가장 좋은 방법은 구축하려는 응용 프로그램의 종류에 따라 다릅니다.

이 경우에는, 보안이 중요합니다 (권한이 없는 사람들이 편집하거나 또는 이슈 리포트나 코멘트의 상세 내역을 보기를 원치 않습니다), 그래서 비관적인 접근방법으로 구현하기로 결정했습니다.

간단히, OnUserAuthorize 이벤트 핸들러에서, 기본 구현을 변경할 수 있다는 것을 의미합니다 (valid에 True로 지정된 것을), False로 수정합니다. 유저 롤이 권한 부여된 롤 컬렉션에 있다면 찾아 확인할 수 있습니다 다음과 같이 구현됩니다:

```

if Assigned(AuthorizeEventObject.AuthorizedRoles) then
    for UserRole in AuthorizeEventObject.UserRoles do
        if AuthorizeEventObject.AuthorizedRoles.IndexOf(UserRole) >= 0
            then valid := True;

```

그러나 가끔 클라이언트 쪽의 DataSnap 프록시 클래스의 생성과 같이 명백하게 권한 부여 된 롤이 없는 오퍼레이션도 있습니다. 앞에서 작성한 OnUserAuthorize 구현 방법으로는 이 기능을 사용할 수 없습니다. 그러한 이유로, 상호 보안적인 접근방법을 사용해야 합니다: 유저 롤이 명백하게 사용하지 못하는 롤의 목록에서 발견되지 않는 한 명백하게 권한이 있는 롤이 지정되지 않았다면 작업이 허용된다고 추측할 수 있습니다, (실제 권한이 있는 롤 이 없을 때, Denied Roles도 역시 거의 없을 것입니다).

다음은 OnUserAuthorize 이벤트 핸들러의 최종 구현부분입니다:

```

procedure TWebModule1.DSAAuthenticationManager1UserAuthorize(Sender: TObject;
    AuthorizeEventObject: TDSAAuthorizeEventObject; var valid: Boolean);

```

```

var
  UserRole: String;
begin
  if Assigned(AuthorizeEventObject.AuthorizedRoles) then
    valid := False // assume NOT OK, unless explicitly allowed !!
  else valid := True; // no Authorized Roles?

  if not valid then
    if Assigned(AuthorizeEventObject.AuthorizedRoles) then
      for UserRole in AuthorizeEventObject.UserRoles do
        if AuthorizeEventObject.AuthorizedRoles.IndexOf(UserRole) >= 0
          then valid := True;

  if valid then
    if Assigned(AuthorizeEventObject.DeniedRoles) then
      for UserRole in AuthorizeEventObject.UserRoles do
        if AuthorizeEventObject.DeniedRoles.IndexOf(UserRole) >= 0
          then valid := False;
end;

```

위의 구현은 또한 유저 룰이 권한이 부여된 룰과 권한이 없는 룰 양쪽에 포함되는지 보기 위해 확실하게 체크한다는 점에 주목하십시오 (이 경우 요청된 서버 메소드에 대해 허용되지 않고 끝납니다).

실제 룰에 의한 권한 지정은 서버 메소드 단에서 해야 할 필요가 있습니다 admin 은 새로운 유저를 추가할 수 있는 권한이 있으며, 관리자는 읽기 전용에서 모든 이슈에 대해서 볼 수 있으며, 감수자는 새로운 이슈들을 볼 수 있으며, 개발자와 테스터들은 이슈에 대한 코멘트를 추가 할 수 있습니다 (이슈들을 해결하기 작업하기 때문에).

클라이언트용 서버 메소드

DirtServerMethods 유닛으로 넘어와서, TServerMethods1 클래스를 서버 메소드를 구현하기 위하여 TDSServerModule로부터 파생하여 확장 할 수 있습니다. 여기에 추가하고 싶은 첫 번째 메소드는 옵션과 활성화 되거나 또는 비활성화된 동작을 지정하는 권한을 허용함으로써 주로 클라이언트 어플리케이션을 지원합니다 유저 룰에 의해서, 유저는 예를 들어 단지 모든 이슈들에 대해 볼 수 있거나 또는 특정 이슈만 편집 할 수 있습니다(한 가지 또는 다른 방법으로 유저에 속한).

이런 목적을 위해, GetCurrentUserID 와 GetUserRoles인 두 개의 간단한 서버 메소드를 구현했습니다.

```

public
  { Public declarations }
  function GetCurrentUserID: Integer;
  function GetUserRoles: String;

```

GetCurrentUserID는 새로운 이슈 리포트를 추가 할 때 또는 기존의 이슈에 주석을 달고 싶을 때 UserID가 필요한 경우에 유용하게 사용됩니다. GetUserRoles는 현재 유저가 속한 룰을 반환 하기 때문에 매우 유용합니다. 클라이언트 쪽에서 기능을 열어서 사용할 수 있습니다.

GetCurrentUserRoles의 구현은 현재 세션으로부터 UserRoles.Text를 반환합니다. 다음과 같이 TDSSessionManager.GetThreadSession를 호출하여 얻습니다

```
function TServerMethods1.GetCurrentUserRoles: String;
begin
  Result := TDSSESSIONMANAGER.GetThreadSession.UserRoles.Text;
end;
```

UserRoles.Text은 단일 문자열에 제로, 하나 또는 그 이상의 롤을 반환할 것입니다. 하나 이상의 롤을 갖고 있다면 선택적으로 CRLF 쌍으로 구분 되어집니다.

GetCurrentUserID 서버 메소드 구현은 약간 더 복잡합니다. 같은 세션을 사용하여 'UserID' 필드의 값(유저가 성공적인 로그 인을 했을 때 할당된 값)을 리턴 하기 위하여 GetData 메소드를 사용합니다. UserID 필드가 존재하지 않거나 유효한 값이 아니라면, -1을 대신 반환 합니다.

```
function TServerMethods1.GetCurrentUserID: Integer;
begin
  Result := StrToIntDef(
    TDSSESSIONMANAGER.GetThreadSession.GetData('UserID'), -1);
end;
```

지금은 UserID 값을 사용하지 않습니다만, 유저들의 리스트를 반환하거나 새로운 유저를 등록 하는 등 매우 흥미로운 주제를 제공합니다.

유저명을 구하기 위한 서버 메소드

새로운 이슈를 추가 할 때 특별히 다른 유저에게 이슈를 할당할 때, 클라이언트 어플리케이션이 유저명 리스트를 (그리고 관련된 UserID 값) 볼 수 있다면 매우 유용할 것입니다. 이러한 목적으로, TServerMethods1 클래스에 유저명과 UserID 값을 갖는 데이터셋을 반환하는 GetUserNames라 불리는 세 번째 서버 메소드를 추가합니다.

이 서버 메소드는 감수자와 개발자 롤을 갖는 유저에 의해서만 호출되는데 이 유저들은 새로운 이슈를 보고 할 수 있거나 기존의 보고된 이슈에 주석 처리만 할 수 있습니다. 결과적으로, 사용자 정의 속성 TRoleAuth을 사용한 선언은 아래와 같이 명백하게 권한이 부여된 롤을 지정합니다:

```
[TRoleAuth('Tester, Developer')]
function GetUserNames: TDataSet;
```

사용자 정의 속성 TRoleAuth에 선택적으로, 서버 메소드가 명백히 금지되었다는 롤을 지정하는 두 번째 아규먼트를 가질 수 있는데 그러나 여기서는 비관적인 권한부여 방식으로 구현했기 때문에 필요 하지 않습니다 명백히 허용된 롤을 가지고 있는 유저만이 이 서버 메소드를 실행 할 수 있을 것입니다.

이 메소드의 구현을 위해서 TSQLConnection 컴포넌트가 필요할 뿐만 아니라 sqlUser라 불리는 TSQLDataSet 컴포넌트가 필요합니다. TSQLDataSet는 TSQLConnection 컴포넌트에 연결하기 위한 SQLConnection 속성이 있으며, 그 속성 안에 CommandType 속성을 Dbx.SQL로 지정합니다. CommandText 에는 사용자 테이블로부터 UserID 와 이름을 가져오기 위한 SQL SELECT 문을 다음과 같이 구성합니다:

```
SELECT UserID, Name FROM [User]
```

이 SQL SELECT 문이 작성되면, sqlUser 테이블을 열고 첫 번째 레코드를 가리키는 GetUserNames

서버 메소드를 구현할 수 있습니다 (이 경우 이미 오픈 되어 있지만 처음에는 연결되어 있지 않음).

```
function TServerMethods1.GetUserNames: TDataSet;
begin
  try
    sqlUser.Open;
    sqlUser.First;
    Result := sqlUser
  except
    on E: Exception do
      begin
        Result := nil;
        sqlUser.Close;
        SQLConnection1.Close
      end
    end;
end;
```

TSQLDataSet을 오픈 할 때 묵시적으로 오픈 되기 때문에 TSQLConnection 컴포넌트는 오픈 할 필요가 없습니다. 함수가 끝나기 전에 확실히 sqlUser을 닫아야 합니다. 왜냐하면 이 서버 메소드의 함수 결과로 오픈 된 TSQLDataSet이 넘어가기 때문입니다.

기존의 유저를 구해오는 것은 좋으나 처음에는 아무런 유저가 없을 것 입니다. 데이터베이스에 특정 패스워드와 룰을 가지고 새로운 유저를 등록하기 위한 서버 메소드가 필요합니다.

새로운 유저를 추가하기 위한 서버 메소드

룰을 가지고 있는 유저만이 새로운 유저를 추가할 수 있습니다. TServerMethods1 클래스에 AddUser로 불리는 4번째 서버 메소드를 추가하며 사용자 정의 TRoleAuth 속성을 사용하여 Admin 룰 만이 이 서버 메소드를 호출할 수 있다고 명백하게 다시 지정하여 구현합니다:

```
[TRoleAuth('Admin')]
procedure AddUser(const User, Password, Role, Email: String);
```

AddUser 프로시저의 매개변수들은 새로운 유저 이름, 패스워드 (암호화 되지 않고 AddUser 서버 메소드 자체에서 해쉬 되는, 이 것을 자유롭게 변경하고, 이미 해쉬 된 패스워드에 넘길 수 있지만), 룰과 메일 주소입니다.

AddUser 메소드는 DIRT 데이터베이스에 연결되는 TSQLConnection 컴포넌트가 필요합니다. 서버 메소드 유닛에 컴포넌트가 있기 때문에, AddUser 서버 메소드를 다음과 같이 구현 할 수 있습니다:

```
procedure TServerMethods1.AddUser(const User, Password, Role, Email: String);
var
  SQLQuery: TSQLQuery;
begin
  SQLQuery := TSQLQuery.Create(nil);
  SQLQuery.SQLConnection := SQLConnection1;
  SQLQuery.CommandText := 'INSERT INTO [User] ' +
    ' (Name, PasswordHASH, Role, Email) ' +
    ' VALUES (:Name, :PasswordHASH, :Role, :Email)';
  SQLQuery.ParamByName('Name').AsString := User;
  SQLQuery.ParamByName('PasswordHASH').AsString := HashMD5(Password);
  SQLQuery.ParamByName('Role').AsString := Role;
```

```

SQLQuery.ParamByName('Email').AsString := Email;
SQLConnection1.Open;
try
    SQLQuery.ExecSQL;
finally
    SQLConnection1.Close;
end;
end;

```

HashMD5 함수는 IdHashMessageDigest 유닛에 있는 인디 TIdHashMessageDigest5 타입을 사용하여 다음과 같이 구현되어 있는 TServerMethods1 클래스에서만 사용할 수 있는 함수입니다:

```

function TServerMethods1.HashMD5(const Str: String): String;
var
    MD5: TIdHashMessageDigest5;
begin
    MD5 := TIdHashMessageDigest5.Create;
    try
        Result := LowerCase(MD5.HashStringAsHex(Str, TEncoding.UTF8))
    finally
        MD5.Free
    end
end;

```

서버 메소드로 MD5 해싱을 공유하고 싶지 않다면 서버 메소드로 확실하게 노출되지 않게 하기 위해서 private (또는 protected) HashMD5 함수로 작성해야 한다는 점에 유의하십시오

결과적으로, 데이터베이스에서 다른 유저를 추가할 수 있도록 관리자 역할을 갖는 적어도 하나의 유저가 필요 한다는 점에 주목하십시오 (수동으로 추가 되어야 하는 첫 번째 유저이거나 관리자 권한이 있는 유저가 유저를 추가할 수 있도록 정의하는 명백한 사용자 지정 속성 없이, 그렇지 않다면 닭이 먼저냐? 달걀이 먼저냐? 하는 문제에 직면하고 처음에 새로운 유저를 추가하기 위한 로그인을 하지 못할 것입니다).

모든 이슈를 구해오기 위한 서버 메소드

이 모든 관리 후, 몇 가지 중요한 기능적인 일을 해야 할 마지막 시간입니다. 이 경우, 보고된 모든 이슈들의 리스트를 표시합니다. 필터를 사용하여, MinStatus 와 MaxStatus 값 사이의 이슈들을 반환하여 클라이언트에서 그리드나 다른 데이터-연결 컨트롤과 연결할 수 있는 데이터셋을 리턴합니다.

5번째 메소드는 GetIssues로 불리는 관리자 룰을 가진 유저들만 사용할 수 있는 함수이며 다음 같이 정의합니다:

```

[TRoleAuth('Manager')]
// Return all issues (read-only) between MinStatus..MaxStatus
function GetIssues(MinStatus,MaxStatus: Integer): TDataSet;

```

이 서버 메소드의 구현을 위해, TSQLConnection 컴포넌트뿐만 아니라 sqlReports라 불리는 TSQLDataSet 컴포넌트가 필요합니다.

TSQLDataSet TSQLConnection 컴포넌트와 연결하는 SQLConnection 속성이 있고 그 속성의 하위 CommandType 속성을 Dbx.SQL로 지정합니다. CommandText는 아래와 같이 Report로부터 모든 필드를 구해오고, WHERE 절에서 Status 필드에 대한 최소값과 최대값의 범위를 지정하는 SQL SELECT문을 구성합니다:

```
SELECT "ReportID", "Project", "Version", "Module", "IssueType",
  "Priority", "Status", "ReportDate", "ReporterID", "AssignedTo",
  "Summary", "Report"
FROM "Report"
WHERE Status >= :MinStatus AND Status <= :MaxStatus
ORDER BY Status
```

이 간단한 SQL 문 사용시 단점은 클라이언트 쪽에서 필드 값 중 일부를 수정할 필요가 있다는 것입니다. IssueType, Priority 와 Status에 대해서는 큰 문제가 없습니다만 ReporterID 와 AssignedTo 값은, UserID 값에 대체할 사용자 이름 리스트가 있어야 합니다. 이 리스트는 테스터나 개발자 롤을 갖는 유저만이 GetUserNames를 호출하여 구할 수 있습니다.

SQL 쿼리가 어찌하든 읽기-전용이 되어야 하기 때문에, 다음과 같이 이러한 유저들의 실제 이름과 ReporterID 와 AssignedTo 필드가 매치되는 유저 테이블과 조인하여 확장할 수 있습니다:

```
SELECT "ReportID", "Project", "Version", "Module", "IssueType",
  "Priority", "Status", "ReportDate",
  UReporterID.Name AS "Reporter", UAssignedTO.Name AS Assigned,
  "Summary", "Report"
FROM "Report"
LEFT OUTER JOIN [User] UReporterID ON UReporterID.UserID = ReporterID
LEFT OUTER JOIN [User] UAssignedTO ON UAssignedTO.UserID = AssignedTO
WHERE Status >= :MinStatus AND Status <= :MaxStatus
ORDER BY Status
```

유저 테이블의 AssignedTo 필드가 반드시 입력되는 필드가 아니기 때문에 LEFT OUTER JOIN 명령어를 사용했다는 것을 주목해 주십시오 (예를 들어 널 값 일수도 있고 그 경우 정상적인 LEFT JOIN은 값을 구하지 못합니다).

이 TSQLDataSet을 사용하여, MinStatus 와 MaxStatus 정수형 매개변수 값을 MinStatus 와 MaxStatus 쿼리 파라미터에 넘겨주는 서버 메소드 GetIssues를 다음과 같이 구현합니다:

```
function TServerMethods1.GetIssues (MinStatus,MaxStatus: Integer): TDataSet;
begin
  try
    SQLConnection1.Open;
    sqlReports.Close;
    sqlReports.ParamByName ('MinStatus').Value := MinStatus;
    sqlReports.ParamByName ('MaxStatus').Value := MaxStatus;
    sqlReports.Open;
    Result := sqlReports
  except
    on E: Exception do
      begin
        Result := nil;
        sqlReports.Close;
        SQLConnection1.Close
      end
    end;
end;
end;
```

함수가 실행되는 동안 TSQLDataSet이 오픈 되어 저 있도록 sqlReports를 닫으면 안 된다는 것을

주목하십시오. 클라이언트 측에서, 결과 데이터셋으로 작업을 할 것입니다.

새로운 이슈를 리포트하는 서버 메소드

매니저 권한을 갖는 유저가 보고된 모든 이슈들을 읽기-전용으로 사용할 수 있는 반면, 감수자 롤을 갖는 유저는 실제 새로운 이슈를 제출할 수 있습니다. 이 경우 아래와 같이 선언된 6번째 서버 메소드, ReportNewIssue가 필요합니다:

```
[TRoleAuth('Tester')]
function ReportNewIssue(Project, Version, Module: String;
IssueType, Priority: Integer; // Status = 1
Summary, Report: String;
AssignedTo: Integer = -1): Boolean;
```

ReportID (고유값 또는 자동 증가 필드), ReporterID와 Status 필드를 제외한 리포트 테이블의 모든 필드들을 넘깁니다. ReporterID는 UserID로 로그인 된 서버 메소드에 의해 결정되며 status 필드는 "보고완료"용으로 자동으로 1 값을 구해옵니다. AssignedTo 필드는 디폴트 값 -1을 갖는 선택적 필드입니다. 이 값은 리포트 된 이슈들이 아직 유저에게 할당되지 않았다는 것을 의미합니다. (AssignedTo 필드와 파라미터가 있거나 없거나)AssignedTo값에 따라, SQL INSERT 문이 약간 달라질 수 있습니다

ReportNewIssue 서버 메소드의 구현은 다음과 같습니다:

```
function TServerMethods1.ReportNewIssue(Project, Version, Module: String;
IssueType, Priority: Integer; // Status = 1
Summary, Report: String;
AssignedTo: Integer = -1): Boolean;
var
  InsertSQL: TSQLQuery;
begin
  Result := False;
  InsertSQL := TSQLQuery.Create(nil);
  try
    InsertSQL.SQLConnection := SQLConnection1;
    if AssignedTo >= 0 then
      InsertSQL.CommandText := 'INSERT INTO [Report] ([Project], ' +
        '[Version], [Module], [IssueType], [Priority], [Status], ' +
        '[ReportDate], [ReporterID], [AssignedTo], [Summary], [Report]) ' +
        'VALUES (:Project, :Version, :Module, :IssueType, :Priority, ' +
        '1, @Date, :ReporterID, :AssignedTo, :Summary, :Report) '
    else // No AssignedTo
      InsertSQL.CommandText := 'INSERT INTO [Report] ([Project], ' +
        '[Version], [Module], [IssueType], [Priority], [Status], ' +
        '[ReportDate], [ReporterID], [Summary], [Report]) ' +
        'VALUES (:Project, :Version, :Module, :IssueType, :Priority, ' +
        '1, @Date, :ReporterID, :Summary, :Report) '

    try
      InsertSQL.ParamByName('Project').AsString := Project;
      InsertSQL.ParamByName('Version').AsString := Version;
      InsertSQL.ParamByName('Module').AsString := Module;
      InsertSQL.ParamByName('IssueType').AsInteger := issueType;
      InsertSQL.ParamByName('Priority').AsInteger := Priority;
      InsertSQL.ParamByName('ReporterID').AsInteger :=
        StrToIntDef(TDSSessionManager.GetThreadSession.GetData('UserID'), 0);
      if AssignedTo >= 0 then
        InsertSQL.ParamByName('AssignedTo').AsInteger := AssignedTo;
```

```

    InsertSQL.ParamByName('Summary').AsString := Summary;
    InsertSQL.ParamByName('Report').AsString := Report;
    Result := InsertSQL.ExecSQL = 1
  except
    on E: Exception do
      CodeSite.SendException(E)
  end
  finally
    InsertSQL.Free;
    SQLConnection1.Close
  end;
end;

```

이 함수는 새로운 레코드가 올바르게 입력된 경우 True를 반환 합니다. 오류 처리는 이 경우에 CodeSite에 예외를 단지 보내는 것으로만 구성되어 있다는 점을 주목하십시오. 필요한 오류 처리를 추가하거나 예외를 재 시도 하는 것은 자유롭게 구현하셔도 좋습니다.

정보를 가져오기 위하여 (GetCurrentUserID, GetCurrentUserRoles, GetUserNames, GetIssues) 또는 새로운 유저를 추가하기 위한 (AddUser) 또는 리포트 하기 위한(ReportNewIssue)등 6개 미만의 서버 메소드를 추가했습니다. 두 개의 메소드 GetUserNames 와 GetIssues는 데이터셋을 반환하는데, 수정하지 못하는 일기-전용 데이터셋 입니다.

개발자 권한을 갖는 유저가 보고서를 검색하고 코멘트를 추가할 수 있도록 하기 위해서, 서버 메소드에 기능을 추가해야 합니다.

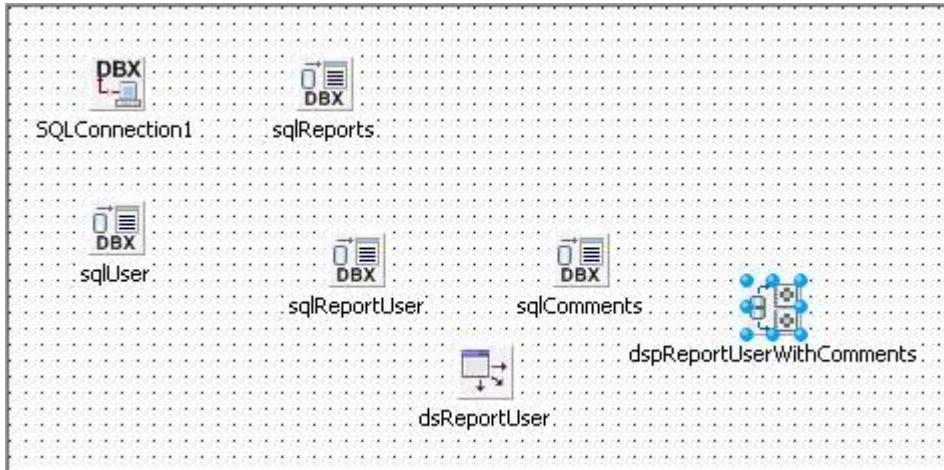
데이터 보내기: 오픈 이슈

개발자들에 대해서 보고된 이슈들을 조회뿐만 아니라 편집하기 위해서, (등록, 수정, 삭제)등 데이터를 변경하기 위해서 클라이언트에서 사용되는 TDataSetProvider를 익스포트해야 하며 서버에 수정 사항들을 다시 적용해야 합니다.

이슈 리포트를 가지고 작업하는 개발자는 초기의 이슈 리포트뿐만 아니라, 이슈에 대해 표시된 모든 코멘트를 보아야 합니다. 결과적으로, TDataSetProvider는 마스터-상세 관계에서Reports 테이블의 레코드뿐만 아니라, Comments 테이블의 레코드도 표시해야 합니다.

이를 구현하기 위해, sqlReportUser 와 sqlComments 라는 이름의 두 개의 TSQLDataSet 컴포넌트를 새롭게 추가해야 하며, dsReportUser라 불리는 TDataSource 컴포넌트 하나를 추가하고 DataSet 속성을 sqlReportUser로 연결하기도 하고 교대로 sqlComments 컴포넌트와 연결하기도 하여 사용합니다.

결과적으로, dspReportUserWithComments 라 불리는 TDataSetProvider 컴포넌트가 필요한데 DataSet 속성을 마스터 sqlReportUser 데이터셋으로 연결합니다.

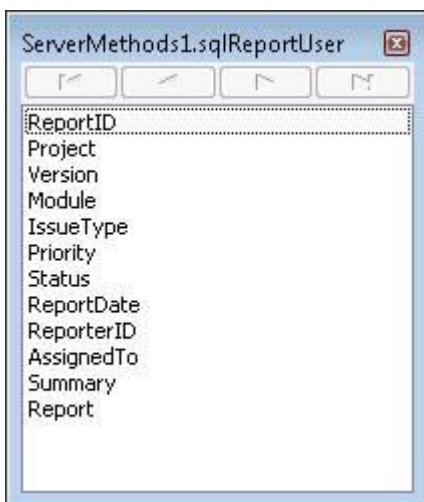


두 개의 TSQLDataSet 컴포넌트의 SQLConnection 속성이 TSQLConnection 컴포넌트와 연결되어 있어야 합니다. sqlReportUser의 SQL 속성은 아래와 같이 정의하며, Status값이 지정한 최소값과 최대값 범위에 있으며 ReporterID 또는 AssignedTo 필드가 현재 유저와 같은 Report 테이블의 레코드를 검색합니다:

```
SELECT "ReportID", "Project", "Version", "Module", "IssueType",
      "Priority", "Status", "ReportDate", "ReporterID", "AssignedTo",
      "Summary", "Report"
FROM "Report"
WHERE (Status >= :MinStatus) AND (Status <= :MaxStatus)
      AND ((AssignedTo = :AssignedTo) OR (ReporterID = :ReporterID))
```

현재 유저에 의해 보고된 문제와 현재 유저에게 할당된 문제 모두 반환된다는 점에 유의하십시오. 이들은 모두 현재 유저에 해당되는 이슈로 간주됩니다.

이 SQL 문은 12 개의 필드를 반환 합니다. 필드 에디터를 시작하기 위해 sqlReportUser 컴포넌트에서 더블 클릭합니다. 그 다음, 필드 에디터에서 더블 클릭하여 "Add all fields"를 선택합니다. 쿼리로 생성한 12개의 필드들의 리스트를 보여줍니다:



이러 필드들의 대부분은 특별한 처리가 필요합니다. 예를 들어 ReportID 필드는 자동 증가 필드입니다. 이 필드의 ProviderFlags 속성을 수정해야 합니다. 하위 속성인 pfInUpdate가 False로

지정되어야 하며(왜냐하면 이 프라이머리 키에 초기값을 할당할 수 없고 어떤 방법으로도 업데이트가 안 되기 때문에), pfInKey는 True로 설정되어 있어야 합니다. pfInWhere는 이미 True로 설정 되어 있습니다.

SQL 문에 4 개의 파라미터: MinStatus, MaxStatus, AssignedTo 와 ReporterID가 있습니다. 그러나 그 중 2 개는 클라이언트에서 제공되며, 2 개는 서버 자체에서 자동으로 채워집니다 (그리고 결과적으로 리스트에서 삭제 되어야 합니다).



MinStatus 와 MaxStatus 입력용 매개변수는 ftInteger 타입이고 리스트에 남아 있어야 합니다. AssignedTo 와 ReportedID 매개변수는 Params 콜렉션에서 지워져야 합니다, 그래서 클라이언트 쪽에서는 보이지 않을 것 입니다 (그리고 서버 쪽에서 자체적으로 현재 사용자가 사용하는 세션에서 UserID 값에 기초한 값으로 채워집니다).

상세 쪽으로 옮겨와서: sqlComments 데이터셋은 DataSource 속성이 마스터 sqlReportUser와 연결 되어 있는 dsReportUser로 연결합니다. 상세 코멘트의 실제 SQL 문은 아래와 같습니다 ReportID 파라미터를 사용하여 리포트에 코멘트를 연결합니다:

```
SELECT CommentID, ReportID, UserID, CommentDate, Comment
FROM Comment
WHERE (ReportID = :ReportID)
```

이 경우, 프라이머리 키 CommentID의 ProviderFlags를 다시 수정해야 합니다. 필드에디터를 실행하기 위해서 sqlComments 컴포넌트에서 더블 클릭하여, 필드에디터에서 더블 클릭하여 "Add all fields"를 선택합니다.



오브젝트 인스펙터에서 CommentID 필드를 선택하여, CommentID 필드의 ProviderFlags 속성을 확장합니다. 하위 속성 pfInUpdate를 False로, pfInKey를 True로 (pfInWhere는 이미 True로 설정 되어 있어야 합니다) 설정합니다. 이렇게 하는 것은 프라이머리 키 CommentID가 값 지정을 위해 서버로 전달되지 않는다는 것을 확실히 하며 레코드를 찾기 위해 WHERE 절에서 사용됩니다.

sqlComments 데이터셋은 상세 sqlReportUser 데이터셋에 연결되어 있기 때문에, TDataSetProvider 는 중첩된 데이터 셋으로 불리는 것처럼, 중첩된 상세 데이터셋으로 마스터와 코멘트(있다면)로 있는 보고서와 함께 둘 다 내보낼 것입니다. 클라이언트 어플리케이션에서 더 상세하게 살펴 보도록 하겠습니다.

수정하고 싶은 TDataSetProvider인 dspReportUserWithComments의 속성인 UpdateMode가 있습니다: 기본 값으로, 이 속성은 upWhereAll로 지정되어 있습니다. 이 값은 UPDATE 와 DELETE 문에서 모든 필드들이 현재 레코드를 찾기 위한 WHERE 절의 부분이 될 수 있다는 것을 의미합니다(Blob 필드만 제외하고). 이 값은 특별히 테이블에 프라이머리 키가 없는 경우에 사용합니다. 그러나 우리 경우와 같이, 프라이머리 키가 있을 때, upWhereAll를 좀 더 효과적인 upWhereChanged로 변경 할 수 있습니다. 이렇게 설정하면, WHERE절이 프라이머리 키뿐만 아니라 변경된 필드들을 포함합니다(수정되지 않고 남아 있는 필드는 포함하지 않습니다). 이것이 수정된 이슈 리포트에서 - 동시에 두 사람이 같은 필드를 수정하지 않는 한 실제로 레코드를 통합할 수 있도록 해줍니다.

마지막으로, upWhereKeyOnly 옵션이 있는데, UPDATE 와 DELETE 문의 WHERE 절에서 다른 유저가 실행한 어떠한 변경 내용도 무시하고 프라이머리 키를 단순히 넘겨주기만 하기 때문에 이 옵션은 위험하다는 점에 주목해 주십시오.

두 개의 데이터셋이 모두 읽기-전용 단 방향 데이터셋이기 때문에, 미스터-상세 관계를 허용하기 위해 한 가지 마지막 작업이 더 필요합니다. 마스터 결과 데이터셋을 통해 진행하는 동안, 마스터 레코드셋에서 다음 레코드로 이동할 때 마다, 상세내용을 "리셋" 해야 합니다. 상세 내용 또한 단 방향 데이터 셋이기 때문에 자동으로 발생하지 않습니다. sqlReportUser의 AfterScroll 이벤트에서 sqlComments 데이터셋을 닫고 다시 오픈 하여, ReportID 파라미터의 새로운 값을 넘깁니다.

다음은 구현 부분입니다:

```
procedure TServerMethods1.AS_sqlReportUserAfterScroll(DataSet: TDataSet);
begin
  sqlComments.Close;
  sqlComments.ParamByName('ReportID').AsInteger :=
    sqlReportUser.FieldByName('ReportID').AsInteger;
  sqlComments.Open;
end;
```

이것은 마스터 레코드를 스크롤 하자마자 각 마스터 레코드에 대한 상세 쿼리를 재 실행합니다, 클라이언트에 TDataSetProvider에 의해 내보내기 전에 서버 쪽에 전체 중첩된 데이터셋을 채웁니다.

이벤트가 디폴트 이름인 sqlReportUserAfterScroll이 아닌 AS_sqlReportUserAfterScroll로 이름 지어진 것에 주목하십시오. 이렇게 한 것은 목적이 있습니다. 그 이유는 다음과 같습니다: 서버 메소드 클래스의 퍼블릭 메소드는 노출이 됩니다. 이것은 퍼블릭 이벤트 핸들러도 마찬가지입니다. 그러나, 이 규칙에 한 가지 예외가 있는데 AS_ 접두사로 시작하는 메소드들은 서버 메소드 클래스로부터 노출되지 않고 숨겨집니다. 이벤트 핸들러를 숨기기 위해 이 기술을 사용할 수 있으며, 클라이언트 쪽에서 서버 메소드 프록시 유닛에 표시 되지 않을 것입니다. (어찌하든 클라이언트에서 호출되지 않기 때문에).

클라이언트 쪽에서 마스터 쿼리의 MinStatus 와 MaxStatus 파라미터의 값을 제공하는 반면, AssignedTo 와 ReporterID 파라미터의 값은 서버 쪽 자체에서 입력해야 합니다. TDataSetProvider의 BeforeGetRecords 이벤트 핸들러 구현으로 이 것이 실행됩니다. 여기에서, 오픈 되기 전에 TSQLDataSet sqlReportUser의 파라미터에 값을 줍니다. 매개변수 ReporterID 와 AssignedTo값에 로그인 후에 유저 right에 대해 현재 쓰레드에 저장된 현재 유저용 UserID값을 지정할 수 있습니다. 구현은 아래와 같습니다:

```
procedure TServerMethods1.AS_dspReportUserWithCommentsBeforeGetRecords(
  Sender: TObject; var OwnerData: OleVariant);
var
  UserID: Integer;
begin
  UserID := StrToIntDef(TDSSessionManager.GetThreadSession.GetData('UserID'), 0);
  if sqlReportUser.Params.FindParam('ReporterID') = nil then
    begin
      with sqlReportUser.Params.AddParameter do
        begin
          DataType := ftInteger;
          ParamType := ptInput;
          Name := 'ReporterID';
          AsInteger := UserID
        end
      end
    else
      sqlReportUser.Params.FindParam('ReporterID').AsInteger := UserID;

  if sqlReportUser.Params.FindParam('AssignedTo') = nil then
    begin
      with sqlReportUser.Params.AddParameter do
        begin
          DataType := ftInteger;
          ParamType := ptInput;
          Name := 'AssignedTo';
          AsInteger := UserID
        end
      end
    end
```

```

end
else
  sqlReportUser.Params.FindParam('AssignedTo').AsInteger := UserID;
end;

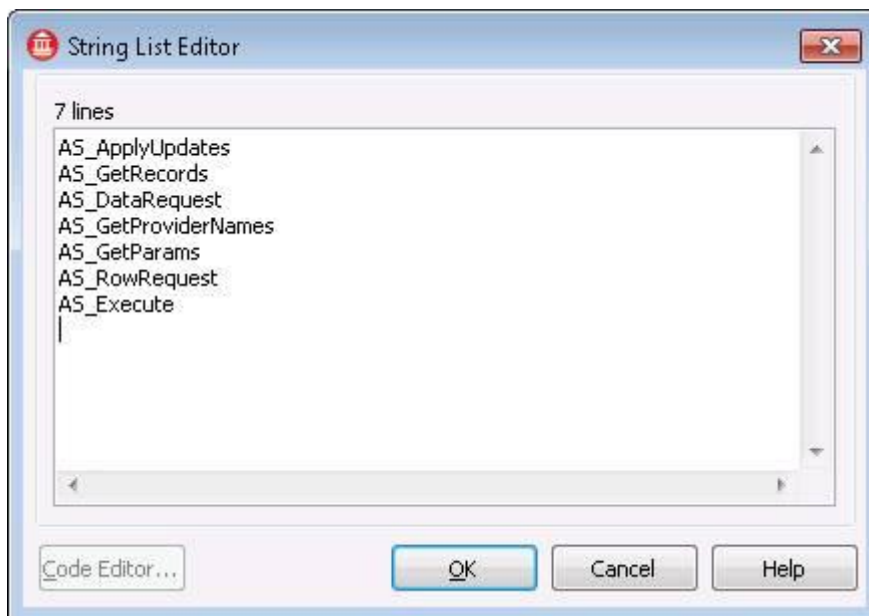
```

sqlReportUser의 모든 파라미터들이 쿼리가 오픈 되기 전에 값이 들어 가야 합니다, 결과적으로, dspReportUserWithComments로부터 리포트와 코멘트 레코드를 요청하는 어떠한 유저도 현재 사용자가 어느 보고자 인지 (예를 들면 ReporterID) 또는 어느 유저에게 할당 되었는지 (예를 들면 AssignedTo), 또는 두 가지 모두의 리포트를 얻을 것입니다. 그러나 다른 사람에 속한 리포트가 아닌 자기의 리포트만 볼 수 있습니다. 그러기 위해, 관리자 권한을 가져야 하며 보고된 모든 이슈들의 개요를 반환하는 (그러나 코멘트없이). GetIssues 서버 메소드를 호출합니다.

TDataSetProvider 룰에 기초한 권한부여

DataSnap 서버에서 마지막으로 TDataSetProvider에 대한 룰 기반 권한 부여 한 가지가 남아있습니다. 서버 메소드와 다르게, 서버 모듈의 컴포넌트에 사용자 지정 속성을 추가 할 수 없습니다. 대신, DirtWebMod로 전환해야 하며 TDSAuthenticationManager 컴포넌트를 사용합니다.

이 컴포넌트는 ApplyTo, AuthorizedRoles 와 DeniedRoles 속성들을 사용하여 Role의 컬렉션을 지정할 수 있는 Roles라는 속성이 있습니다. ApplyTo 속성은 매소드 이름, 클래스 이름을 지정 할 수 있는 곳입니다 클래스 이름은 메소드 이름 다음에 마침표를 입력합니다. 여기서는 미리 정의해놓은 7개의 IAppServerFastCall 메소드들의 호출을 허용하는 개발자 유저만 할 수 있습니다. Apply 속성에 다음과 같이 7개를 추가합니다:



특별한 서버 메소드 클래스로부터 7개의 메소드를 단지 보호하기 원하다 는 표시를 위해 TServerMethods1에 접두사를 사용할 수 있다는 점을 주목하십시오. Apply 속성이 지정되었기 때문에, AuthorizedRoles 속성에 대한 룰을 개발자 권한을 가지고 있는 유저만이 TDataSetProvider를 사용할 수 있도록 지정하는 Developer로 입력합니다.

단순히 개발자가 기존의 이슈 리포트를 편집하거나 코멘트를 추가하도록 허락하는 것이 약간 제한적이라고 느끼실 수 있습니다. 이 리스트에 관리자뿐만 아니라 자유롭게 감수자를 추가합니다 (비록 그것이 관리자에게 달려있지만, 보고서를 수정하고 코멘트를 추가 할 수 있는 능력을 유저에게 주고 싶다면).

서버 배포

DataSnap D.I.R.T. 서버가 안전하고 확실하게 배포되기 위해서는, HTTPS 프로토콜을 사용하는 마이크로소프트의 인터넷 정보 서비스에 ISAPI DLL를 배포하거나, 전송 채널을 암호화 하기 위한 RSA 와 PC1 필터들을 사용하는 스탠드-어론 DataSnap 서버를 배포해야만 합니다. 다음 번에는, TCP/IP가 속도가 많이 빠르기 때문에 통신 프로토콜로 HTTP가 아닌 TCP/IP를 사용할 것을 권합니다. 우리가 사용하였던 배포 옵션은 실제 다음과 같습니다:

- IIS상에서 HTTPS를 사용한 ISAPI DLL
- PC1 와 RSA 필터 기능이 있는 TCP/IP를 사용한 단독 실행 파일

IIS 자동 로드 발란싱, 재활용(recycling), 그리고 서버의 CPU 와 메모리 사용제한 등을 지원하는 응용프로그램 풀을 구성할 수 있도록 해주는 추가적인 혜택이 있습니다. 이러한 이유로, D.I.R.T. DataSnap 서버는 도메인에 설치된 SSL 인증서가 있는 웹 서버의 IIS에 ISAPI DLL로 배포됩니다. 이 기술 백서의 목적을 위해 www.bobswart.nl 사용합니다. 가상 디렉터리 DataSnapXE가 생성되었으며, 특수 어플리케이션 풀이 이 가상 디렉터리에 있는 어플리케이션용으로 구성됩니다. DirtServer.dll은 이 가상 디렉터리 안에 위치하게 되고, 결과는 아래 URL 입니다.

<https://www.bobswart.nl/DataSnapXE/DirtServer.dll>

이 URL 을 호출하는 것은 "DataSnap XE Server" 텍스트를 단지 표시하지만, 적어도 서버가 배포되었는지 충분히 확인합니다. 물론, DirtServer.dll이 데이터베이스에 연결되는지 확인해야 합니다. 그런 목적으로, 요청된 데이터베이스 드라이버 또한 서버 머신에 배포되어야 합니다. SQL Server 2008용으로, 검색 경로에 (Windows\System32 디렉터리와 같이, DLL 이 로드 할 수 있도록)위치해야 하는 dbxmss.dll을 의미합니다.

DirtServer.dll 과 the dbxmss.dll과는 별개로 웹 서버에 다른 어떠한 파일도 배포할 필요가 없습니다. 프로젝트의 uses 절에 서버 프로젝트 안에서 자체적으로 MIDAS.dll를 링크하는 MidasLib 유닛을 추가 할 수 있기 때문에, MIDAS.dll은 필요하지 않습니다.

TCP/IP 와 RSA 와 PC1 필터를 사용한 DataSnap 스탠드어론 서버를 배포한다면, 두 개의 특정 SSL DLLs: libeay32.dll 과 ssleay32.dll 또한 배포해야 합니다 -서버 머신에 이미 확실히 설치 되어야 합니다. 이러한 DLL들은 RSA 필터 (PC1 필터에 사용되는 패스워드를 암호화하는)사용을 위해 필요합니다. 두 개의 DLL이 없다면, 서버에 연결하고자 하는 클라이언트는 "연결이 닫혔습니다"라는 메시지가 표시 되는데, 서버가 PC1 키나 기타 등등을 암호화하기 위한 RSA 필터를 시작하기 위한 두 개의 DLL을 로드 할 수 없기 때문입니다.

RSA 와 PC1 필터를 사용한 TCP/IP 서버로 연결하든 또는 HTTPS를 사용한 ISAPI 필터에 연결되든 클라이언트에서는 어찌하든 두 개의 DLL은 필요합니다. 우선 클라이언트를 먼저 작성하고 배포에 관한 부분은 다음에 하도록 하겠습니다.

DATA SNAP 클라이언트

배포된 D.I.R.T. DataSnap 서버를 테스트할 수 있는 DataSnap 클라이언트를 작성해야 합니다. 이 클라이언트는 USB 장치에 저장 하여 어떠한 윈도우 환경의 머신(DirtServer 프로젝트에 접근하기 위해 <https://www.bobswart.nl> 서버에 인터넷 연결이 되는)에도 설치되고, 실행 할 수 있는 간단한 단독 실행 파일입니다.

새로운 VCL 폼 어플리케이션을 생성하여, DataSnap 서버에 데이터 액세스 컴포넌트를 집중화 하기 위해 데이터 모듈을 추가합니다. 처음 단계로, 데이터 모듈에 TSQLConnection 컴포넌트를 배치해야 합니다. Driver 속성을 Datasnap로 지정하고, 오브젝트 인스펙터에서 DataSnap의 특정 속성들을 보기 위해 Driver 항목을 확장합니다. CommunicationProtocol은 https로, Port는 443로 그리고 HostName은 www.bobswart.nl (물론 자유롭게 원하는 서버를 연결하셔도 됩니다)로 설정합니다. URLPath 속성은 웹 서버상의 DirtServer의 위치를 지정하는 /DataSnapXE/DirtServer.dll로 설정합니다. LoginPrompt 속성은 False로 지정한다는 것을 잊지 마십시오 서버에 연결하고자 할 때 로그인 대화상자가 나타난다면 필요 없습니다 그 이유는 클라이언트에서 서버로 해쉬 된 패스워드 때문에 (실제 패스워드 대신에)를 넘겨야 하기 때문입니다

여기서 한 가지 주목할 사항은 DSAuthUser 와 DSAuthPassword 속성에 적당한 값을 지정하지 않으면, 서버에 접속할 수 있으나 결과적으로 DataSnap 클라이언트 클래스의 생성은 되지 않는다는 점입니다. 내 컴퓨터에 있는DirtServer는 문제가 있을 수 있습니다 (하지만 원한다면 프로젝트 아카이브에서 미리 생성된 클라이언트 클래스 유닛 DBXClientClasses.pas를 사용할 수 있습니다). 작성한 DataSnap 서버용으로, 물론 OnUserAuthenticate 이벤트 핸들러를 일시적으로 비활성화 할 수 있으며, 클라이언트 클래스를 생성할 필요가 있는 만큼 valid 에 True를 할당하고, 그리고 인증 체크를 다시 활성화합니다.

클라이언트 클래스가 생성되었기 때문에 다음과 같은 퍼블릭 서버 메소드들로 구성되어 있는 TServerMethods1Client 클래스를 확인하실 수 있습니다:

```

type
  TServerMethods1Client = class(TDAdminClient)
  private
    ....
  public
    constructor Create(ADBXConnection: TDBXConnection); overload;
    constructor Create(ADBXConnection: TDBXConnection;
      AInstanceOwner: Boolean); overload;
    destructor Destroy; override;

    function GetCurrentUserID: Integer;
    function GetCurrentUserRoles: string;
    function GetUserNames: TDataSet;
    procedure AddUser(User: string; Password: string;
      Role: string; Email: string);
  
```

```

function ReportNewIssue(Project: string; Version: string;
  Module: string; IssueType: Integer; Priority: Integer;
  Summary: string; Report: string; AssignedTo: Integer): Boolean;
function GetIssues (MinStatus: Integer; MaxStatus: Integer): TDataSet;
end;

```

그러나 TServerMethods1Client의 인스턴스를 생성하고 서버 메소드를 호출하기 전에, 먼저 서버와 연결을 해야만 합니다.

로그인

다음의 예제는 프로젝트에 새로운 폼을 추가하고 로그인 폼을 디자인합니다:



패스워드를 입력하는 TEdit (edPassword라 불리는)의 PasswordChar 속성을 지정합니다.

이 대화로그는 모달 창으로 표시되며, 입력한 유저 이름과 패스워드를 TSQLConnection의 파라미터에 할당되어 추가하거나 기존의 값을 대체합니다. 서버 어플리케이션에서와 같은 방법으로 패스워드를 해쉬 해야 하며, 다음은 로그인 이벤트에 대한 코드입니다:

```

procedure TFormClient.Login1Click(Sender: TObject);
var
  MD5: TIdHashMessageDigest5;
  Server: TServerMethods1Client;
  UserRoles: String;

begin
  DataModule1.SQLConnection1.Connected := False;
  UserID := -1;

  with TFormLogin.Create(Self) do
    try
      if ShowModal = mrOK then
        begin
          SQLConnection1.Params.Values['DSAuthenticationUser'] := Username;
          MD5 := TIdHashMessageDigest5.Create;
          try
            SQLConnection1.Params.Values['DSAuthenticationPassword'] :=
              LowerCase(MD5.HashStringAsHex(Password, TEncoding.UTF8));

            SQLConnection1.Connected := True; // try to login...
            Server := TServerMethods1Client.Create(SQLConnection1.DBXConnection);
            try
              UserID := Server.GetCurrentUserID;
              UserRoles := Server.GetCurrentUserRoles;

              // Adjust GUI capabilities based on UserRoles...

            finally

```

```

        Server.Free
    end
    finally
        MD5.Free
    end
end
finally
    Free // TFormLogin
end
end;

```

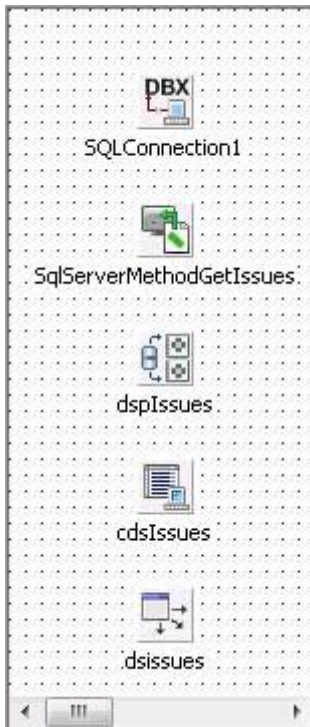
유저 룰에 기초하여 GUI 일부를 숨기거나 비활성화 하는 실제 코드는 보여주지 않을 것입니다. 이것은 독자들을 위해 연습으로 남겨 놓겠습니다. 물론 예제인 DirtServer 와 DirtCleaner 클라이언트 프로젝트를 다운로드 하여 소스 코드를 더 상세히 살펴 보실 수 있습니다.

로그인 폼은 기존의 DataSnap 서버와의 연결을 끊고 TSQLConnection 컴포넌트의 DSAAuthenticationUser 와 DSAAuthenticationPassword 파라미터에 새로운 값을 할당합니다. 로그인이 성공하든 못하든 서버와의 연결이 다시 된다면, 다음에 GetCurrentUserID 와 GetUserRoles 서버 메소드를 호출합니다.

데이터모듈(DATA MODULE)과 서버 메소드

데이터모듈의 TSQLConnection를 사용하여 DataSnap 클라이언트 클래스를 이미 생성하였지만 실습에서 서버 메소드들을 직접 호출 할 필요가 없었습니다. 로그인 코드는 이미 두 가지의 GetCurrentUserID와 GetUserRoles 호출을 보여주었지만, 다른 서버 메소드들은 전형적으로 읽기 전용 혹은 업데이트 가능한 데이터셋을 검색하기 위해 사용됩니다 그 중 예외인 서버 메소드 ReportNewIssue는 되에서 다루겠습니다.

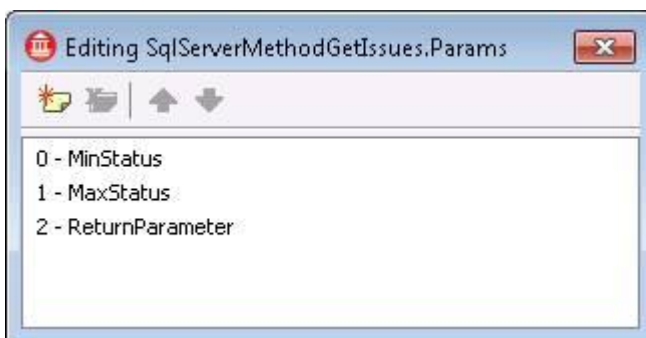
자 이제 데이터모듈로 돌아와서 추가적으로 4개의 컴포넌트를 내려놓겠습니다: SqlServerMethodGetIssues로 불리는 TSqlServerMethod 컴포넌트, dspIssues라 불리는 TDataSetProvider 컴포넌트 cdsIssues라 불리는 TClientDataSet 컴포넌트, 마지막으로 dsIssues가 이름인 TDataSource 컴포넌트



SqlServerMethodGetIssues 컴포넌트는 SqlConnection1 컴포넌트의 SqlConnection 속성을 연결하고 DataSnap 서버 Server와 통신합니다. 실행하고자 하는 서버 메소드를 선택하기 위해 ServerMethodName의 드롭-다운 리스트를 사용합니다. 서버 서버 메소드들의 리스트를 표시하기 위해 TSqlConnection 컴포넌트는 반드시 실행 중인 DataSnap 서버에 연결되어 있어야 한다는 점에 주목하십시오

예제의 경우, TServerMethods1.GetIssues 서버 메소드를 호출 해야 합니다. 이 메소드는 “관리자” 권한을 가지고 있는 유저만 호출할 수 있으며 모든 이슈를(상태 값의 특정한 범위 내에서) 가지고 있는 읽기-전용 데이터셋을 반환합니다

드롭-다운 리스트에서 이 서버 메소드를 선택하자마자, SqlServerMethodGetIssues의 Params 속성을 확인 할 수 있습니다. MinStatus, MaxStatus 와 ReturnParameter (하나의 데이터셋)등 3개의 파라미터입니다 :



MinStatus 와 MaxStatus 파라미터의 값은 디폴트 값으로 지정하거나, 코드에서 값을 지정할 수 있습니다

이름이 dspIssues인 TDataSetProvider는 DataSet 속성을 SqlServerMethodGetIssues로 연결해야 합니다.

다음으로 cdsIssues TClientDataSet의 ProviderName 속성을 dspIssues로 연결해야 합니다. 데이터셋의 결과값이 수정 될 수 없기 때문에, cdsIssues의 ReadOnly 속성을 True로 설정하는 것이 좋은 방법이며, 그래서 이 데이터셋에 연결되어 있는 어떠한 데이터-컨트롤도 사용자가 어떤 변경도 하지 못하도록 할 것입니다.

마지막으로, TDataSource 컴포넌트인 dsIssues는 DataSet 속성을 TClientDataSet 컴포넌트 cdsIssues로 지정해야만 dsIssues 데이터 인식 컨트롤과 연결 할 수 있습니다.

이제 클라이언트 메인 폼에서, TDBGridReports가 이름인 TDBGrid를 위치시키고 다른 서버 메소드나 이 메소드를 사용하여 정보를 표시합니다. MinStatus 와 MaxStatus 필터의 명백한 값을 넘겨주고 GetIssues 서버 메소드의 결과를 표시하기 위해서, 다음과 같은 코드를 작성합니다:

```

procedure TFormClient.ViewAllIssues1Click(Sender: TObject);
// Manager - all reports (read-only)
begin
  DBGridReports.DataSource := nil;
  DataModule1.SQLConnection1.Connected := True;
  try
    DataModule1.cdsIssues.Close;
    DataModule1.SqlServerMethodGetIssues.Params.
      ParamByName('MinStatus').AsInteger := MinStatus;
    DataModule1.SqlServerMethodGetIssues.Params.
      ParamByName('MaxStatus').AsInteger := MaxStatus;

    DataModule1.cdsIssues.Open;
    DBGridReports.DataSource := DataModule1.dsIssues;
  finally
    DataModule1.SQLConnection1.Connected := False
  end
end;

```

데이터모듈의 TSQLConnection 컴포넌트를 사용하여 DataSnap 서버의 연결을 시작한다는 것에 주목하십시오. SqlServerMethodGetIssues 파라미터들의 값을 주고, 서버 메소드를 호출하여 datasource를 DBGridReport의 DataSource에 지정하여 결과 값을 볼 수 있으며, 결과적으로 DataSnap 서버의 연결을 다시 닫습니다. 비록 서버 연결을 계속 유지할 수 있지만, 요청 후에 바로 연결을 끊는 것이 클라이언트가 좀 더 강력해 질것입니다 (항상 새로운 연결로 시작합니다) 항상 새롭게 연결해야 하기 때문에 약간 느려질 수 있으나, 단지 활성화된 연결만 유지되어야 하는 것에 비해 서버는 좀 더 확장성이 증가합니다.

결과 데이터셋은 정수형으로 표시되는 IssueType, Priority 와 Status같은 필드들을 포함합니다. 이러한 정수 값들을 우리가 인식하기 쉬운 문자 값으로 매핑 할 수 있습니다. 이렇게 하기 위해, 데이터모듈의 cdsIssues TClientDataSet에서 더블 클릭하여 필드에디터를 시작합니다. 필드에디터에서 오른쪽 마우스를 클릭하여 "Add all fields"를 선택합니다.

IssueType, Priority 와 Status 필드들에 대해서 간단한 정수 값을 인식하기 쉬운 스트링 값으로 변환하기 위해서 다음과 같이 OnGetText 이벤트 핸들러를 작성합니다:

```

procedure TDataModule1.IssueTypeGetText(Sender: TField;
var Text: string; DisplayText: Boolean);
begin
  case Sender.AsInteger of
    1..4: Text := 'Minor ' + Sender.AsString;
    5..8: Text := 'Major ' + Sender.AsString;
    9..10: Text := 'Critical ' + Sender.AsString;
  end;
end;

procedure TDataModule1.PriorityGetText(Sender: TField;
var Text: string; DisplayText: Boolean);
begin
  case Sender.AsInteger of
    1..3: Text := 'Low ' + Sender.AsString;
    4..7: Text := 'Medium ' + Sender.AsString;
    8..10: Text := 'High ' + Sender.AsString;
  end;
end;

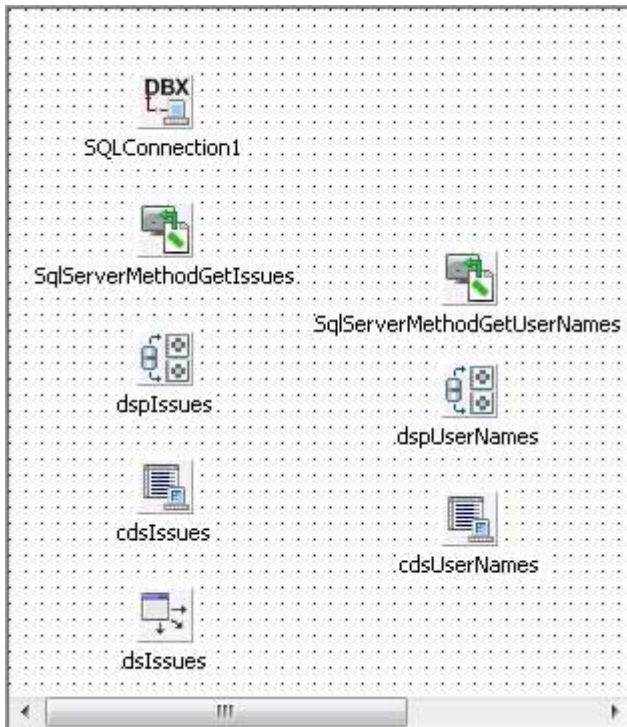
procedure TDataModule1.StatusGetText(Sender: TField; var Text: string;
  DisplayText: Boolean);
begin
  case Sender.AsInteger of
    1: Text := 'Reported';
    2: Text := 'Assigned';
    3: Text := 'Opened';
    6: Text := 'Solved';
    7: Text := 'Tested';
    8: Text := 'Deployed';
    10: Text := 'Closed';
  end;
end;

```

이러한 필드들을 자유롭게 변경하여도 되지만 이 정도면 충분합니다. 원래의 정수 값은 IssueType 과 Priority 필드 값에 여전히 표시될 뿐만 아니라, 실제 값이 무엇이었는지 알려준다는 것에 주목하십시오

유저명과 서버 메소드

비슷한 방법으로, GetUserNames 서버 메소드를 호출하기 위한 목적만이라면 TSqlServerMethod 컴포넌트를 사용할 수 있습니다. 유저 이름과 UserID 값들의 리스트를 표시하기 위해서 cdsUserNames라는 이름을 갖는 TDataSetProvider 컴포넌트와 cdsUserNames라는 이름을 갖는 TClientDataSet 컴포넌트를 매려 놓습니다.



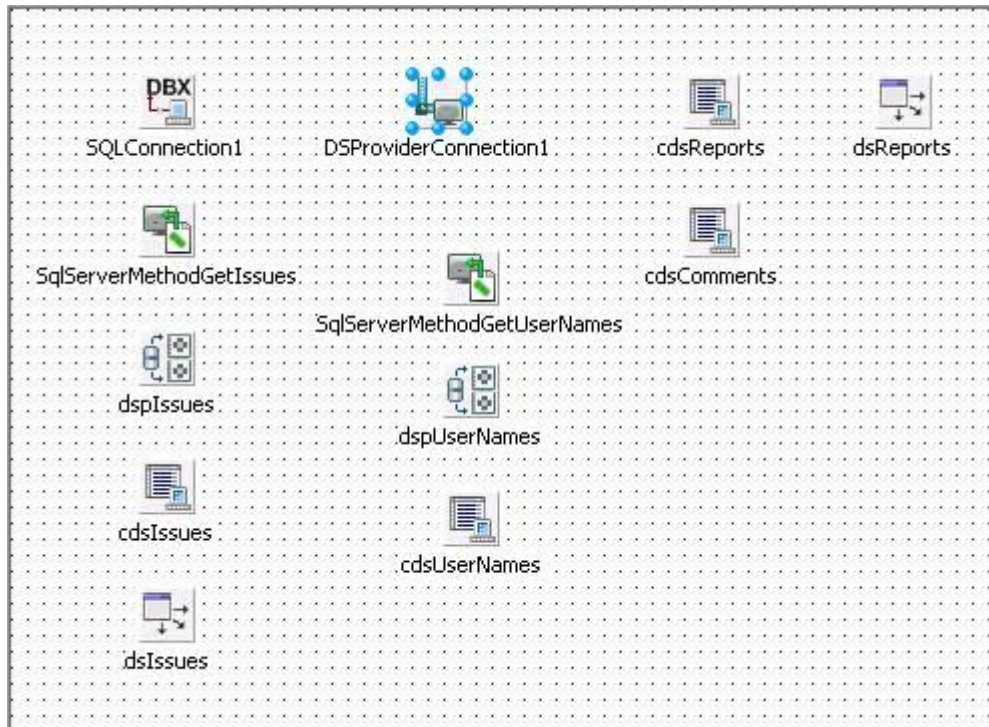
SqlServerMethodGetUserNames는 TSQLConnection 컴포넌트에 연결되어 있어야 하며 TServerMethods1.GetUserNames 서버 메소드와 연결합니다. cdsUserNames는 모든 레코드들을 구해오는 cdsUserName로 불리는 TClientDataSet에서 사용되는 TDataSetProvider와 연결됩니다.

필드 에디터를 사용하여, cdsUserNames가 Name 과 UserID 필드 두 개가 있다는 것을 확인하실 수 있습니다. 이 데이터셋은 이름의 목록을 제공하거나 UserID 값을 변경하기 위한 "지원" 데이터셋으로 사용되어 집니다 (예를 들면 AssignedTo 또는 ReportedID 필드로부터).

DSPROVIDERCONNECTION

이제는 개발자 또는 테스트 역할의 유저들을 위한 리포트 된 이슈들로 화제로 돌려보겠습니다. 그런 경우에는, GetIssues 서버 메소드를 호출하지 않고 서버부터 dspReportUserWithComments라 불리는 TDataSetProvider를 사용해야 합니다

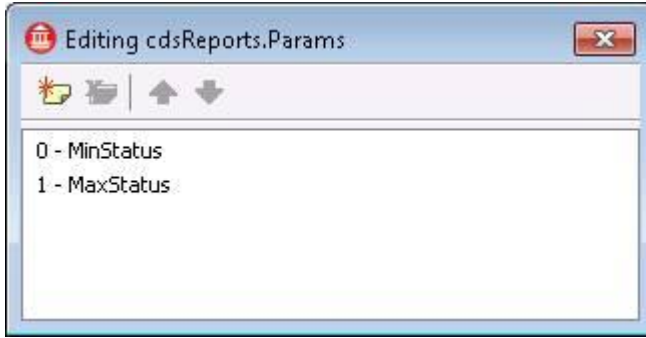
이를 위해 데이터모듈에, cdsReports 와 cdsComments로 이름을 갖는 2개의 TClientDataSets와 cdsReports에 연결되는 dsReports라 불리는 하나의 TDataSource 컴포넌트뿐만 아니라 TDSProviderConnection 컴포넌트를 추가해야 합니다.



DSPProviderConnection 컴포넌트와 DataSnap 서버 메소드 클래스를 연결합니다. 우리의 경우 TServerMethods1 하나의 클래스만 존재하기 하기 때문에 DSPProviderConnection 컴포넌트의 SQLConnection 속성을 TSQLConnection로 연결하고, ServerClassName 속성에서 서버 메소드 클래스의 정확한 이름을 지정해야 합니다. 이름은 TServerMethods1입니다 (명백히, 만일 호출되는 서버 메소드가 같다면 확인할 수 있습니다).

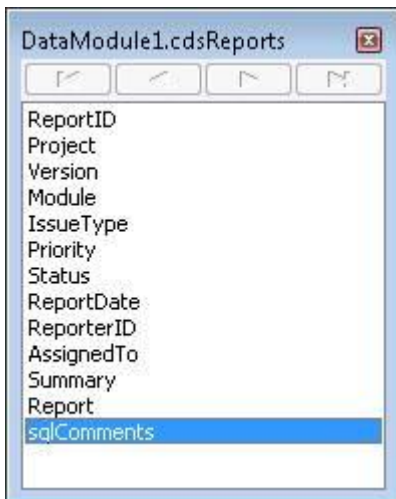
다음 단계로, cdsReports TClientDataSet 컴포넌트를 cdsReports의 RemoteServer 속성을 지정하여 DSPProviderConnection와 연결해야 합니다. 모든 것이 올바르게 설정되면, DataSnap 서버는 실행되고 연결되어, cdsReports의 ProviderName 속성 선택을 위한 드롭-다운 리스트를 열어서 dspReportUserWithComments를 선택합니다.

결과적으로, dspReportUserWithComments에 의해 보내진 전체 마스터-상세 데이터셋은 cdsReports에 포함 되어 질것입니다. 파라미터 뿐만 아니라 필드의 설정이 필요합니다. 우선, cdsReports상에서 오른쪽 마우스를 클릭하여 Fetch Params를 선택합니다. 이 작업은 cdsReports의 Params 콜렉션에 MinStatus 와 MaxStatus 파라미터를 가져와서 표시합니다 (서버의 파라미터 콜렉션에서 지운 AssignedTo 와 ReportedID 파라미터는 없음).



이러한 파라미터에 디폴트 값을 지정할 있으나, 코드에서 간단하게 명백한 값을 지정하도록 하겠습니다.

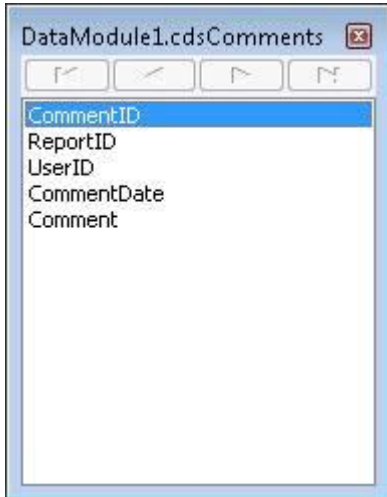
필드 에디터를 실행하기 위하여 cdsReports에서 더블 클릭합니다. 필드 에디터에서 오른쪽 마우스를 클릭하여 "Add all fields"를 선택합니다. 다음은 끝에 sqlComments라는 특별한 필드를 갖는 13개 필드 리스트 결과입니다



이 특별한 필드는 이 경우에 "comment" 레코드들인 마스터 리포트 레코드에 속한 연결된 상세 레코드들을 포함합니다. cdsComments라 불리는 두 번째 TClientDataSet를 추적하기 위해서 TDataSetField 타입인 이 필드를 사용할 수 있습니다.

오브젝트 인스펙터를 사용하여, cdsComments인 TClientDataSet의 DataSetField 속성에 cdsReportssqlComments를 지정합니다. cdsComments 데이터셋은 cdsReports 데이터셋에서 현재 마스터 레코드에 대한 상세 레코드들을 자동으로 포함합니다.

이름이 cdsComments인 TClientDataSet 에서 더블 클릭하여 필드에디터를 시작하고 오른쪽 마우스를 클릭하여 "Add all fields"를 선택합니다.



TClientDataSets은 첫 번째 필드와 프라이머리 키에 관련하여 약간의 수정이 필요합니다. ReportID (리포트 테이블에서) 와 CommentID (코멘트 테이블에서)를 자동 증가 값을 갖는 필드로 지정하였기 때문에, 데이터베이스에 이 값들을 지정하지 않아도 됩니다. DBMS 자체가 이 값들을 생성합니다. 결과적으로, cdsReportsReportID 필드와 cdsCommentsCommentID 필드에 대해서 ProviderFlags속성을 편집하는데 pfInUpdate 하위 속성을 False로 pfInKey 하위 속성을 True로 설정해야 합니다. 이 것은 TClientDataSets가 프라이머리 키가 실제 키임을 인식하고 서버에서 수정되거나 입력되지 않는다는 것을 확실히 해줍니다.

참조 필드(LOOKUP FIELDS)추가

cdsReports로 돌아가서 필드 리스트를 다시 살펴보겠습니다. 관리자 유저를 구해오던 cdsIssues 에서 이전에 했던 것과 같이 몇 개의 필드들이 수정되어야 합니다. 이러한 필드들은 IssueType, Priority 와 Status 이며 앞에서 보여진 것처럼 같은 OnGetText 이벤트 핸들러를 공유할 수 있습니다 (같은 데이터 모듈에 있습니다).

유저 이름 없이 단지 UserID 값을 가지고 있는 ReportedID 와 AssignedTo 필드 역시 수정이 필요합니다. cdsReports에 ReporterName 과 AssignedName인 두 개의 참조 필드를 추가하기 위하여 참조 소스로 cdsUserNames 테이블을 사용합니다. 필드에디터에서 오른쪽 마우스를 클릭하여 "New field..."를 선택하여 새로운 참조 필드를 정의합니다.다음은 ReporterName에 대한 예제입니다:

cdsReports의 AssignedTo 필드를 위한 AssignedName 참조 필드에 대해서도 같은 작업을 합니다. cdsComments에서 UserName 참조 필드에서 선택 되어지는 UserID 필드도 잊지 마십시오.

자동 증가 필드

특별히 마스터-상세 관계에서 자동 증가 필드 작업을 하기 위해 몇 가지 단계가 더 필요합니다. ReportID 와 CommentID 프라이머리 키에 대한 값을 쓰기 는 할 수 없지만, 새로운 코멘트나 새로운 이슈를 생성하기 할 수 있도록 클라이언트 쪽에서 값을 할당해야만 합니다. 이런 새로운 레코드들을 위해, -1부터 값을 감소해가면서 음수 키 값을 사용해야 합니다 두 개의 TClientDataSet의 AfterInsert 이벤트를 사용하여 다음과 같이 구현합니다:

```

procedure TDataModule1.cdsReportsAfterInsert (DataSet: TDataSet);
begin
  cdsReportsReportID.AsInteger := -1;
end;

procedure TDataModule1.cdsReportsOrCommentsAfterPostOrDelete (DataSet: TDataSet);
begin
  cdsReports.ApplyUpdates (-1)
end;

```

여기의 코드는 단순히 -1을 할당하고 더 이상 값을 감소 시키지 않는다는 점에 주목하십시오. 클라이언트 어플리케이션이 post 나 delete후에 바로 서버에 모든 업데이트 내용을 보내기 때문에 수를 감소할 필요가 없습니다. 가능한 음수 키 값을 갖는 레코드(현재 에디터 되고 있는)가 하나 이상이 될 수 없다는 것을 의미합니다. (입력이나 수정 후에)post 바로 후에, 새로운 레코드는 서버로 바로 보내지고, 서버로부터 실제 자동 증가 값을 구해오기 위해서는 cdsReports와 cdsComments의 내용을 리프레쉬 해야 합니다.

그런 목적으로, TClientDataSet 둘 다 AfterDelete 와 AfterPost 이벤트 핸들러를 구현해야 합니다. 다행히 여기 모든 경우가 해야 할 일이 같기 때문에 4 개의 모든 이벤트 핸들러는 하나로 공유할 수 있습니다:

```

procedure TDataModule1.cdsReportsOrCommentsAfterPostOrDelete(DataSet: TDataSet);
begin
  cdsReports.ApplyUpdates(0);
  cdsReports.Refresh
end;

```

ApplyUpdates를 호출하여 서버로 변경된 내용을 보내고, 바로 후에 서버로부터 데이터를 다시 가져오기 위해 Refresh 메소드를 호출합니다.

Refresh는 프라이머리 키 값을 리프레쉬 하는 것은 아니라 선택된 레코드의 데이터들을 리프레쉬 합니다. 이것은 나쁜 아이디어 같을 수 있으나 그 것은 확실히 현명한 생각일 것입니다 레코드 양이 많다면 이 것은 좋지 않지만, 이 경우에는, DataSetProvider는 현재 유저에 속하고 지정한 MinStatus 와 MaxStatus 범위에 속하는 리포트만(코멘트가 있는) 반환합니다, 수천 건의 레코드를 생성한다면 이 유저는 테스트할 때 힘들 것이며 개발자는 너무 많은 작업을 해야 합니다

Developer 나 Tester 권한을 갖는 유저가 이슈들을 보는 것은 다음과 같이 구현합니다:

```

procedure TFormClient.ViewMyIssues1Click(Sender: TObject);
// Developer/Tester personal reports
begin
  DBGridReports.DataSource := nil;
  DataModule1.SQLConnection1.Connected := True;
  try
    DataModule1.cdsReports.Close;
    DataModule1.cdsReports.Params.ParamByName('MinStatus').AsInteger :=
      MinStatus;
    DataModule1.cdsReports.Params.ParamByName('MaxStatus').AsInteger :=
      MaxStatus;
    DataModule1.cdsReports.Open;
    DBGridReports.DataSource := DataModule1.dsReports;

    // nested dataset
    DBGridReports.Columns[DBGridReports.Columns.Count-1].Visible := False;
  finally
    DataModule1.SQLConnection1.Connected := False
  end
end;

```

TDBGrid에서 마지막 칼럼은 중첩된 데이터셋입니다. cdsReport의 필드에디터에서 필드의 Visible 속성을 false로 설정하거나, TDBGrid에서 마지막 칼럼의 Visible 속성을 False로 지정할 수 있습니다. 위의 코드는 후자를 선택한 구현입니다.

중첩된 칼럼과 별도로, 리포트 된 이슈들을 보기 위한 코드는 Manager가 SqlServerMethodGetIssues에 의해 보내진 GetIssues 서버 메소드를 호출 하기 위해 필요한 것과 매우 유사합니다. Manager의 호출 결과는 읽기-전용 데이터셋 (실제로 ReadOnly 속성을 사용하여 읽기-전용으로 설정합니다)이지만, 개발자와 감수자의 호출 결과는 수정 가능한 데이터 셋입니다.

이러한 수정을 위해 사용자가 그리드에서 데이터를 수정하기를 원하지 않기 때문에, 새로운 화면은 개별적인 이슈 보고서에서 편집과 업데이트를 하도록 설계되었습니다.

리포트된 이슈 폼

리포트된 이슈 폼 작성은 (dsComments 와 dsReports로 불리는) 두 개의 TDataSources와 마스터

Report와 상세 코멘트 레코드의 내용을 표시하기 위한 몇 개의 데이터-인식 컨트롤만 내려 놓으면 됩니다. 그러한 목적으로 작성한 화면 레이아웃은 아래와 같으며 물론 자유롭게 레이아웃을 변경하셔도 됩니다:

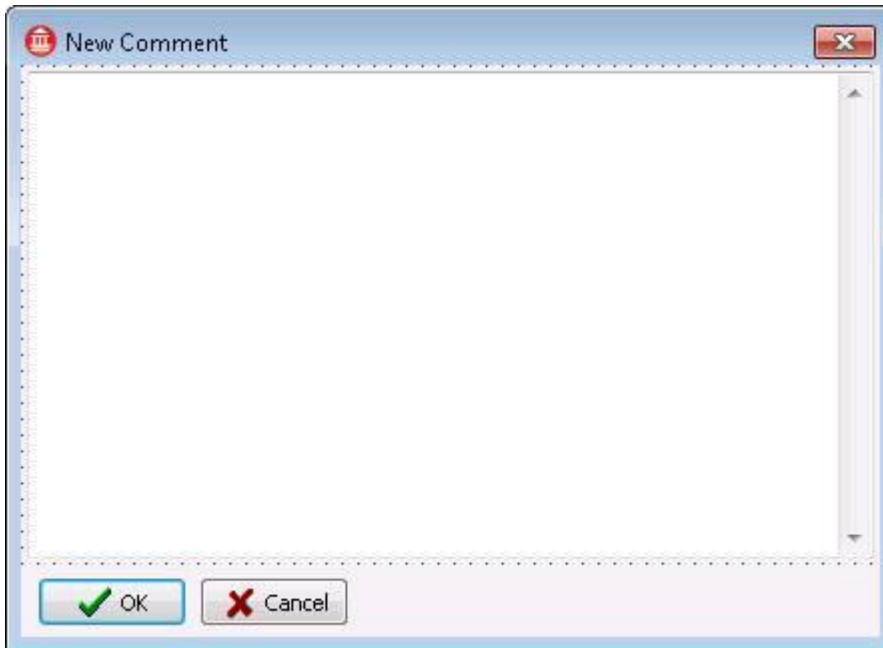
코멘트들은 TDBGrid에 표시되며, 현재 코멘트의 상세내역은 두 번째 TDBMemo에 표시됩니다. 코멘트를 그리드에서 추가하는 것은 안되지만(가능하지만), 새로운 폼에서 추가하는 것이 더 좋습니다. 새로운 코멘트를 입력하고 조회하기 위해 특별한 폼을 다음과 같이 설계하였습니다:

```

procedure TFormIssue.btnNewCommentClick(Sender: TObject);
begin
  with TFormComments.Create(Self) do
    try
      ShowModal;
    finally
      Free;
    end
  end;

```

새로운 폼은 두 개의 버튼과 수직 스크롤바가 활성화된 메모장을 사용한 매우 간단한 형식입니다.



실제 코멘트들은 단지 새로운 코멘트를 입력하기 위해 필요하다는 것에 주목하십시오, 클라이언트 어플리케이션이 코멘트하고 싶은 현재 리포트, 현재 사용자뿐만 아니라 날짜를 알고 있기 때문에, 수동 입력이 필요한 필드는 실제 코멘트 자체뿐입니다.

결과적으로, OK 버튼의 구현은 다음과 같습니다:

```
procedure TFormComments.BitBtn1Click(Sender: TObject);
begin
  DataModule1.cdsComments.Insert;
  DataModule1.cdsComments.CommentID.AsInteger := -1;
  DataModule1.cdsComments.UserID.AsInteger := UserID;
  DataModule1.cdsComments.CommentDate.AsDateTime := Now;
  DataModule1.cdsComments.Comment.AsString := Memo1.Text;
  DataModule1.cdsComments.Post;
  Close;
end;
```

이는 대화상자를 닫고, 리프레쉬된 리포트와 코멘트의 내용을 확인하기 위해 유저에게로 다시 돌아갑니다.

새로운 이슈 보고

조회할 한 가지, 새로운 이슈 보고가 더 남아있습니다: 이 기능을 위해, 프로젝트, 버전, 모듈, IssueType, Priority, Summary, Report 와 선택적으로 AssignedTo 매개변수를 넘겨서 ReportNewIssue 서버 메소드를 수동으로 호출할 수 있습니다. 간단한 입력 컨트롤을 사용하여 서버 메소드를 직접 호출하는 것은 어려운 일은 아닙니다.

그러나, 단순히 cdsReports 테이블에 새로운 레코드를 입력하기 위해서 DSProviderConnection에 연결되어 있는 기존의 cdsReports TClientDataSet를 또한 사용할 수 있는데 프라이머리 키에 -1 이 할당되어 있고(자동적으로), GUI를 디자인하기 위해 데이터-인식 컨트롤을 사용합니다. 그래서

ReportNewIssue 서버 메소드 (어쨌든 감사자 에게만 허용되는)를 무시하고, 감사자와 개발자 모두가 사용 가능한 데이터 셋 프로바이더 기능을 사용합니다.

데이터-인식 리포트 이슈 폼은 데이터모듈의 cdsReports와 연결되는 TDataSource 한 개와 데이터-인식 컨트롤로 구성 되어 다음과 같이 정의 되었습니다.

폼 생성 하는 동안, cdsReports 데이터셋에, ReporerID에 UserID를 할당하고, 보고된 것처럼 Status에 1값을, ReportDate에 현재 날짜로 입력을 할 수 있습니다. ReporterID, Status 와 ReportDate 필드는 이 폼에서 읽기-전용으로 되어 있습니다.

Cancel 버튼은 Cancel 하고 폼을 닫는 반면 OK 버튼은 Post 하고 폼을 닫습니다. 구현은 다음과 같습니다:

```

procedure TFormNewIssue.FormCreate(Sender: TObject);
begin
  dsReports.DataSet.Open;
  dsReports.DataSet.Insert;
  dsReports.DataSet.FieldName('ReporterID').AsInteger := UserID;
  dsReports.DataSet.FieldName('Status').AsInteger := 1;
  dsReports.DataSet.FieldName('ReportDate').AsDateTime := Date;
end;
procedure TFormNewIssue.btnOKClick(Sender: TObject);
begin
  dsReports.DataSet.Post;
  Close
end;
procedure TFormNewIssue.btnCancelClick(Sender: TObject);
begin
  dsReports.DataSet.Cancel;
  Close
end;

```

IssueType, Priority, AssignedTo 필드들은 간단하게 TDBEdit 컴트롤을 사용하여 값을 할당합니다. (1..10까지의 범위로 입력을 제한하는) IssueTypes 와 Priorities를 입력 받기 위해 숫자 spinedit 컨트롤을 사용하고 AssignedTo 필드를 위해 알려진 유저들을 표시하는 드롭-다운 콤보박스를 사용하는 것이 좋은 방법입니다. AssignedTo 필드는 반드시 입력되지 않아도 되기 때문에, (심지어 유저에게 이슈 보고서를 아직 할당하고 싶지 않더라도 항상 선택하도록 하는) TDBLookupComboBox 컴포넌트를 사용하지 않습니다. 그래서 대신에 FormCreate 이벤트에서 유저의 이름을 구해서 채운 일반적인 TCombobox를 사용합니다 다음은 추가된 코드입니다:

```
DataModule1.cdsUserNames.Open; // in case it was closed
DataModule1.cdsUserNames.First;
cbAssignedTo.Items.Clear;
cbAssignedTo.Items.Add('Nobody');

while not DataModule1.cdsUserNames.Eof do
begin
  // current user name
  if DataModule1.cdsUserNames.UserID.AsInteger = UserID then
    edReporter.Text := DataModule1.cdsUserNames.Name.AsString;

  cbAssignedTo.Items.AddObject (
    DataModule1.cdsUserNames.Name.AsString,
    TUserID.Create(DataModule1.cdsUserNames.UserID.AsInteger));

  DataModule1.cdsUserNames.Next;
end;
```

TUserID 클래스는 UserID를 유지하기 위해 생성했던 특별한 지원 클래스입니다 그래서 이 오브젝트를 ComboBox의 Items의 AddObject 메소드의 두 번째 아규먼트로 추가할 수 있습니다

```
type
  TUserID = class
    UserID: Integer;
    constructor Create(const NewUserID: Integer);
  end;

constructor TUserID.Create(const NewUserID: Integer);
begin
  inherited Create;
  UserID := NewUserID
end;
```

메모리 누수를 피하기 위해서, 폼이 닫힐 때 TComboBox로부터 다시 오브젝트를 해제해야 합니다 FormClose에서 다음과 같이 처리됩니다:

```
procedure TFormNewIssue.FormClose(Sender: TObject;
  var Action: TCloseAction);
var
  i: Integer;
begin
  for i:=0 to cbAssignedTo.Items.Count-1 do
    if Assigned(cbAssignedTo.Items.Objects[i]) then
      cbAssignedTo.Items.Objects[i].Free
  end;
```

현재 유저 이름을 이름이 edReporter인 TEdit의 Text 속성에 지정해서 보고서를 제출한 사람의 이름을 볼 수 있습니다 (우리 자신의 이름이어야 합니다).

새로운 이슈 리포트 화면의 디자인 결과는 다음과 같습니다:

IssueType 과 Priority TSpinEdit 컨트롤은 기본값으로 1을 갖고 최소값 1과 최대값 10을 갖습니다. 디폴트 값은 자유롭게 변경할 수 있는데 예를 들어 원한다면 5로 변경하여 쉽게 더 긴급한 이슈로 보고할 수 있습니다.

마지막 선택적인 확장으로, 프로젝트에 있는 기존의 값들의 목록을 검색하여 현재 데이터셋에서 Version 과 Module 입력 필드용 TComboBox 컨트롤을 또한 추가 할 수 있습니다. 그러나 이 부분은 독자들을 위해 마지막 실습으로 남겨 놓겠습니다.

클라이언트 배포

DataSnap 클라이언트 프로젝트의 uses 절에 MidasLib 유닛을 추가한다면, 단독 실행 파일은 거의 더 이상 작업은 필요 없습니다. 데이터베이스 드라이버는 필요 없습니다. 그러나 ISAPI DLL 은 HTTPS를 사용하고 TCP/IP 스탠드-어론 서버는 RSA 와 PC1 필터를 사용하기 때문에, DataSnap 클라이언트와 함께 libeay32.dll 와 ssleay32.dll 파일들을 배포해야 하거나 이미 클라이언트 머신에 있어야 합니다.

DirtCleaner 실행 파일과 두 개의 SSL 지원 DLL들은 DirtServer와 연결하기 위해 인터넷 연결이 되어야 하지만, 그 외에는 필요한 것이 없습니다. 유효한 유저명과 패스워드는 관리자, 개발자 또는 감사자의 롤로 참여하고 서버에 연결하기 위해 필요한 전부입니다. DirtServer 와 DirtCleaner 프로젝트의 소스코드는 다운로드 받으실 수 있습니다. 더 수정된 버전은 제가 작성한 DataSnap XE 교육용 매뉴얼에서 제공되며 설명하겠습니다(등록된 독자들을 위해 교육용 매뉴얼에서 누락 또는 오타/버그 리포트에 사용됩니다).

요약

이 실전 DataSnap XE 기술백서에서, 규모는 작지만 실전의 안전한 DataSnap 서버 어플리케이션을 어떻게 설계하고 구현하고, ISAPI DLL로 마이크로 소프트의 인터넷 정보 서비스에 어떻게 배포하는지 설명하였습니다. 안전성은 보안 통신 채널로 HTTPS (혹은 스탠드-어론 서버용 RSA/PC1 필터) 뿐만 아니라 인증과 권한 부여를 이용하여 구현됩니다. DataSetProviders의 서버 메소드를 호출하기 위하여 클라이언트에서 어떻게 DataSnap 서버와 연결되는지를 보여드렸습니다.

데이터셋 작업 시 다루어지는 기술 이슈 중에서, 어떻게 자동 증가 필드를 사용하여 특별히 미스터-상세와 연관된 데이터셋과 함께 작업하는지 설명하였습니다.

이 어플리케이션은 DataSnap XE의 많은 새로운 특징과 향상된 점 없이는 불가능합니다. 델파이 XE버전의 DataSnap는 COM에 기초한 DataSnap 과 MIDAS의 원래의 버전으로부터 많이 발전되었으며 다시 DataSnap 2010 또는 2009와 비교하여 지속적으로 확장되고 향상되었습니다.

참고자료

실전 RAD Studio – DataSnap 2010 백서

<http://www.embarcadero.com/rad-in-action/datasnap>

델파이 XE DataSnap 개발 교육용 매뉴얼

<http://www.ebob42.com/courseware/>

저자소개

Bob Swart (aka Dr.Bob)는 네덜란드에 본사를 둔 그의 회사 Bob Swart Training & Consultancy (eBob42)의 IT 컨설턴트, 개발자, 리셀러, 저자, 트레이너 그리고 웹 마스터 입니다. Bob은 1993부터 델파이 컴퍼런스에서 강연해 왔습니다. Bob은 몇 가지 도서의 공동 저자이며 다수의 컴퓨터 잡지에 저자로 참여하고 있습니다. 그의 교육용 매뉴얼은 Lulu.com 혹은 그의 웹 사이트를 통해 (이-메일 지원과 무료 업데이트를 포함하는)판매됩니다. Bob은 네덜란드 소프트웨어개발 네트워크의 델파이 섹션을 리드하고 있으며 영국 개발자 그룹의 일원이며 웹 마스터입니다.

Website: <http://www.drBob42.com>

Email: Bob@eBob42.com

LinkedIN: <http://www.linkedin.com/in/drBob42>



엠바카데로 테크놀로지는, 1993 년에 설립한 데이터베이스 툴 제작사입니다. 2008 년에 볼랜드의 개발 툴 부문 「CodeGear」를 합병하였습니다. 현재는 애플리케이션 개발자와 데이터베이스 기술자가 다양한 환경에서 소프트웨어 애플리케이션을 설계, 구축, 실행하기 위한 툴을 제공하는 최대 규모의 독립계 툴 제작사입니다. 미국 기업의 총수입 랭킹 「포천 100」중 90 개 기업과 전세계 300 만 이상의 고객이, 엠바카데로의 Delphi®, C++Builder®, JBuilder® 등 CodeGear™제품과 ER/Studio®, DBArtisan®, RapidSQL® 등 DatabaseGear™ 제품을 채용해, 생산성의 향상과 혁신적인 소프트웨어 개발을 실현하고 있습니다. 엠바카데로 테크놀로지는, 샌프란시스코에 본사를 두고, 세계 각국에 지사를 전개하고 있습니다. 보다 자세한 내용은, <http://www.devgear.co.kr>를 참고하시기 바랍니다.