=== Programlama ve Tasarım ===

--- Delphi 2010 DataSnap ---

--- DataSnap'in Geçmişi ---

Delphi 3'te MIDAS, Delhpi 4'te MIDAS 2, Delphi 5'te MIDAS 3 olarak devam eden bu süreçte COM, TCP/IP ve HTTP tabanlı olarak kuvvetli uzaktan yönetim data modülleri oluşturmak mümkündü. Bu yöntem Delphi 6 ile birlikte DataSnap adını aldı ve Delphi 2007'ye kadar bu framework'e dokunulmadı.

Delphi 2009, COM'a olan bağımlılıkları ortadan kaldırıp, uzak sunucu nesnelerini ve sadece TCP/IP yöntemiyle istemcilerin bağlantılarını çok daha hızlı ve yük bindirmeyecek bir şekilde getirerek, ama aynı zamanda .NET istemcileri için Delphi Prism 2009 kullanma yeteneği getirerek DataSnap'in yeni mimarisini tanıttı.

Delphi 2010 ise DataSnap 2009 mimarisini temel alarak ve yeni hedefler için 2 ayrı sihirbaz bulundurma desteği (VCL Form, Windows Servisi, Konsol ama ISAPI gibi web hedefleri, CGI, Web Uygulama Debugger'ı), HTTP(S) iletişim platformu, HTTP yetkilendirmesi, istemci tarafında geri çağırma fonksiyonları (client callback function), REST ve SPON için destek, sıkıştırma ve kripto (compression and encrpytion) için destek gibi ek özellikler getirerek daha da geliştirdi.

--- DataSnap ıçin Örnek Veri ---

Bu dökümanda size, demo ve örneklerle oynamanız için ısrar edeceğim. Delphi'nin DBX4, ADO, dbGo veya diğer veri erişimi araçlarıyla pek çok farklı veritabanına bağlanması için destek vermesi buradaki örneklerle oynamayı çok kolaylaştıran bir durumdur. Ben veritabanı yönetim sistemi olarak BlackFish veritabanını DBX4 ile kullanacağım ve employee.jds veritabanı dosyası, XP için C:\Documents and Settings\All Users\Documents\RAD Studio\7.0\Demos\database\databases\BlackfishSQL adresinde Vista ve Windows 7 için ise C:\Users\Public\Documents\ RAD Studio\7.0\Demos\database\databases\BlackfishSQL adresinde bulunuyor. Ekran görüntülerinde de görüneceği gibi ben örnekler için işletim sistemi olarak Windows 7 Professional ve DataSnap ISAPI örnekleri geliştirmek için de Windows Server 2008 Web Edition kullanıyorum.

--- DataSnap Windows Hedefleri ---

DataSnap VCL Formları, Windows Servsileri ve Konsol Uygulamaları olmak üzere 2 ayrı Windows hedefine destek vermektedir. Bu bölümde her birinin yararlarını, farklılıklarını ve en iyi kullanılacağı durumları anlatacağım. Örnek bir DataSnap sunucusu oluşturacağım, bununla ilgili olarak TDSServer, TDSServerClass, TDSTCPServerTransport, TDSHTTPService, TDSHTTPWebDispatcher ve TDSHTTPServiceAuthenticationManager gibi belirli sunucu bileşenlerini ve TDSServerModule sınıfını kullanacağım.

TCP/IP ve HTTP gibi farklı iletişim platformlarını bunların etkilerini ve potansiyel yararlarını işleyeceğim. DataSnap sunucu nesnesinin yaşam döngüsü için sunucu, oturum, yürütme (Server, Session, İnvocation) gibi farklı seçenekleri, bunların etkileriyle ve gerçek hayattaki kullanımı için tavsiyeleriyle inceleyeceğim. En sonunda uygulama ile ilgili bazı noktalara açıklık getireceğim.

--- DataSnap Sunucu Örneği ---

Object Repository (New Item dediğimiz zaman Delphi ile ilgili dosyaları oluşturacağımız sihirbaz) üzerinde Windows tabanlı olarak DataSnap Server ve IIS (Internet Bilgi Sistemi - Internet Information System) gibi bir web server tarafından sağlanacak WebBroker tabanlı DataSnap Server (sunucu) projeleri (web projeleri için) olmak üzere 2 ayrı sihirbaz bulunmaktadır. Burada biz ilki ile başlayacağız.

Delphi 2010'u açtığımız ve Dosya Menüsünden Yeni - Diğer dediğimiz zaman (File- New - Other) DataSnap Sunucusu ile ilgili olarak karşımıza 3 tane farklı seçenek çıkacak. DataSnap Sunucusu (DataSnap Server), DataSnap WebBroker Sunucusu (DataSnap WebBroker Server), DataSnap Sunucu Modülü (Server Module). Burada DataSnap Sunucusunu çift tıklama ile seçerek (ilerleyen bölümlerde diğerlerine de göz atacağız.) karşımıza çıkan diyalog penceresine göz atacağız.

Diyaloğun ilk bölümü projenin hedefi ile ilgilidir. Varsayılan olarak ana formu olan bir VCL Form uygulamasını seçeceğiz. İkinci seçenek Konsol uygulamasıdır ki bize komut isteminde bir pencere açar (İstek ve sonuçları takip etmek için ideal yöntemdir, basit writeln komutları ile sunucu uygulamalar içinde ne olduğu gayet açık bir şekilde görülebilir.). İki uygulama da demolar ve başlangıç uygulamaları için idealdir, ama DataSnap sunucu uygulamasını en sonunda yayarken veya uygulamaya geçirirken daha az idealdir. Yeni DataSnap mimarisi COM'a dayanmadığından yeni gelen bir istemci bağlantısı DataSnap sunucu uygulamasına erişemeyecektir. Eğer gelen sunucu bağlantı taleplerini karşılamak istiyorsaki DataSnap sunucusu açılmış ve çalışır bir vaziyette olmalıdır. Ve eğer gelen talepleri 7*24 karşılamak istiyorsak, DataSnap sunucu uygulaması bu süreç içerisinde çalışıyor olmalıdır. VCL Form veya Konsol uygulaması için bunu yaparken, bir hesabın Windows'a bağlanması, DataSnap sunucu uygulamasını çalıştırması gerekir ki bu ideal çözümden uzak bir yaklaşımdır. Bu durumda 3. seçenek olan Windows Servis uygulaması daha ideal bir çözümdür, çünkü birinin makinaya bağlanmasına gerek kalmadan bilgisayar açıldığı zaman otomatik olarak kurulup konfigürsayon ayarlarını otomatik olarak yapıp hizmete başlayabilir. Servis uygulamasının dezavantajı ise, varsayılan olarak kendini masaüstünde göstermemesi ve diğer çözümlere göre daha zor debug edilmesidir. Buna rağmen bu 3 dünyanın en iyi özelliklerini göstermek mümkündür. 1 dakika içerisinde DataSnap Servis uygulaması gibi bir DataSnap VCL Form uygulamasının ve DataSnap Konsol uygulamasını nasıl bir proje grubu olarak oluşturululacağını, aynı belirli sunucu metodlarını nasıl paylaştıklarını, ihityaç duyulduğu zaman tek bir DataSnap sunucu uygulamasının nasıl 3 farklı hedefe (Form, Konsol, Servis uygulaması) derlenebileceğini anlatacağım.

Diyaloğun ikinci bölümü kullanabileceğimiz farklı iletişim protokollerini göstermektedir. DataSnap 2009'la karşılaştırıldığında ek olarak HTTP yetkilendirmesi ile yapılacak bir şekilde HTTP protokolünü de kullanabiliriz. Daha esnek olması için ben buradaki bütün protokolleri kullanmayı öneriyorum, bu şekilde TCP/IP ve HTTP protokollerini beraber kullanabiliriz, ve HTTP ile birleşimi için HTTP Yetkilendirmesini kullanabiliriz.

Diyaloğun son kısmı benim görüşüme göre oldukça iyi bir şekilde konfigüre edilmiştir. Sunucu metod sınıfını oluşturmanın yararlarından bahsederek, eğer istersek ata türler olan TPersistent, TDataModule veya TDSServerModule'ü seçebiliceğimizden

bahseder. Sonuncusu başlangıçtan beri RTTI'ya destek verdiği için en iyi çözümdür. (Eğer düzenli olarak TDataModule kullanıyorsanız, veya hernagi bir dataset kullanmayıp görsel olmayan kontroller kullanıyorsanız, TPersistent da sizin için yeterlidir.)

DSServer.pas dosyasının ufak bir bölümü, bize TDSServerModule ve TDataModule'den türetilmiş TProviderDataModule arasındaki ilişkiyi gösterir.

```
 TDSServerModuleBase = class(TProviderDataModule)
        public
                procedure BeforeDestruction; override;
                destructor Destroy; override;
        end;

{$MethodInfo ON}
TDSServerModule = class(TDSServerModuleBase)
end;
{$MethodInfo OFF}
```

Ne yapacağınızdan emin olmadığınız zamanlarda, TDSServerModule sınıfını ata sınıf olarak seçiniz.

--- Çok Hedefli Proje Grubu - VCL Form ---

Daha önce bahsettiğim gibi çok hedefli bir DataSnap Sunucu uygulaması yazacağız. Bunun için öncelikle diyalogda VCL Forms seçeneğini bütün protokolleri seçerek seçeceğiz.

Diyalogda tamam dediğimiz zaman karşımıza çıkan proje ServerContainerUnit1.pas, ServerMethodsUnit1.pas ve Unit1.pas dosyalarından oluşan, varsayılan olarak Project1.dproj olarak adlandırılan bir projedir. Öncelikle bunu kaydederken Unit1.pas'ı MainForm.pas, ServerContainerUnit1.pas'ı ServerContainerUnitDemo.pas, ServerMethodsUnit1.pas'ı ServerMethodsDemo.pas ve Project1.dproj'u da DataSnapServer.dproj olarak kaydetmeliyiz.

Sonrasında 1 dakika içerisinde bu proje grubunun içerisine Konsol ve Servis projelerini de ekleyeceğiz. Öncelikle ne durumda olduğumuzu anlamak için projeyi derlemeye çalışmalıyız. Projeyi derlediğimiz zaman bir hata mesajı verecek (Bu benim hatam, otomatik oluşturulmuş ServerMethodsUnit1.pas dosyasının adını değiştirdiğim için). Bu mesaj ServerContainerDemo.pas dosyasının içinde ServerMethodsDemo.pas olan dosyanın önceki adının uses kısmında bulunmasından dolayı oluşur (30. satırda). Bu hatayı düzeltmek için dosyanın içeriğine girerek uses kısmında yeni kaydettiğimiz ServerMethodsUnitDemo adını uses kısmına ekleyip hata veren kısmı çıkararak dosyayı kaydedip tekrar derlememiz gerekir. Bu seferki derlemede de 37. satırda ServerMethodsUnit1, TServerMethods1 sınıfında nitelendirici olarak kullanıldığı için hata verir. ServerMethodsUnit1'i ServerMethodsUnitDemo olarak kaydedip projeyi tekrar derlediğimiz zaman hata vermeden çalışacaktır.

ServerContainerUnitDemo dosyasının implementation adı verilen bölümü şu şekildedir.

```
implementation
        uses
        Windows, ServerMethodsUnitDemo;

        {$R *.dfm}
```

```
        procedure TServerContainer1.DSServerClass1GetClass(
        DSServerClass: TDSServerClass; var PersistentClass: TPersistentClass);
        begin
                PersistentClass := ServerMethodsUnitDemo.TServerMethods1;
        end;
end.
```

--- ServerContainerUnitDemo ---

ServerContainerUnitDemo dosyasının dizayn bölümüne baktığımız zaman en az 5 adet bileşen görürüz. Bunlar; TDSServer, TDSServerClass, TDSTCPServerTransport (TCP/IP iletişimi için), TDSHTTPService (HTTP iletişimi için) ve TDSHTTPAuthenticationManager (HTTP yetkiendirmesi için). İlk ikisi her zaman gelir, diğer üçü ise diyalogda işaretlenen iletişim protokol seçeneklerine göre gelir.

--- TDSServer ---

Bu bileşenin AutoStart, HideDSAdmin, Name ve Tag olmak üzere 4 adet özelliği bulunmaktadır. AutoStart özelliği varsayılan olarak True olarak gelir ki, bu DataSnap sunucusunun form oluşturulduğu zaman otomatik olarak başlatılacağını bildirir. Eğer bu değeri True olarak atamazsak, Start (Başlat) metodunu elle çağırmamız gerekir, ve aynı şekilde Stop (Durdur) metodunu da elle çalıştırmamız gerekir. Aynı zamanda Started fonksiyonunu DataSnap Server'ın başlatılıp başlatılmadığını kontrol etmek için çağırabiliriz. HideDSAdmin özelliği varsayılan olarak False gelir. Bunu True yaptığımız zaman, DataSnap sunucusuna bağlanan istemciler, TDSAdmin sınıfındaki sunucu tarafında oluşturulmuş sunucu metodlarını çağıramazlar. TDSAdmin sınıfı tam olarak bir sınıf değildir, ama çağırabildiğimiz TDSAdmin metodları DSNames unit'inde dökümante edilmiştir.

```
        TDSAdminMethods = class
                public
                        const CreateServerClasses = 'DSAdmin.CreateServerClasses';
                        const CreateServerMethods = 'DSAdmin.CreateServerMethods';
                        const FindClasses = 'DSAdmin.FindClasses';
                        const FindMethods = 'DSAdmin.FindMethods';
                        const FindPackages = 'DSAdmin.FindPackages';
                        const GetPlatformName = 'DSAdmin.GetPlatformName';
                        const GetServerClasses = 'DSAdmin.GetServerClasses';
                        const GetServerMethods = 'DSAdmin.GetServerMethods';
                        const GetServerMethodParameters =
'DSAdmin.GetServerMethodParameters';
                        const DropServerClasses = 'DSAdmin.DropServerClasses';
                        const DropServerMethods = 'DSAdmin.DropServerMethods';
                        const GetDatabaseConnectionProperties =
'DSAdmin.GetDatabaseConnectionProperties';
                end;
```

TDSServer bileşenini 5 adet olayı vardır. Bunlar; OnConnect, OnDisconnect, OnError, OnPrepare ve OnTrace'dir. Log dosyasına tek satırlık metin atmak gibi farklı durumlara cevap verebilmek için bu 5 olaya farklı tetikleyiciler yazılabilir.

OnConnect, OnDisconnect, OnError ve OnPrepare olayları DxContent, TransPort, Server ve DbxConnection bileşenlerinin bazı özelliklerini içeren TDSEventObject nesnesinden türetilen bir argüman içerir. OnConnect ve OnDisconnect için kullanılan TDSConnectEventObject türü, ChannelInfo gibi ConnectionProperties özelliği içerir.TDSErrorEventObject türü hataya sebep olan Exception nesnesini, TDSPrepareObjectEvent ise bizim kullanmak istediğimiz MethodAlias ve ServerClass gibi özellikleri içerir.

OnTrace olay tetikleyicisi TDBXTraceInfo türünden bir argüman içerir. OnTrace olay tetikleyicisi oluşturulduğu zaman TDBXTraceInfo ve CBRType gibi bazı türlerin derleyici tarafından tanınmadığına dair hatalar verir. bunu çözmek için uses kısmına DBXCommon (TDBXTraceInfo için) ve DBXCommonTypes (CBRType için) eklenmelidir.

OnConnect olay tetikleyicisi sırasında bağlantı için ChannelInfo hesaplanmalıdır. (Bu bilgiyi log dosyasına yazmak için LogInfo gibi rastgele bir fonksiyon yazılabilir.)

```
procedure TServerContainer1.DSServer1Connect(
DSConnectEventObject: TDSConnectEventObject);
begin
        LogInfo('Connect ' + DSConnectEventObject.ChannelInfo.Info);
end;
```

Sunucunun ne yaptığını iyi bir şekilde görmek için OnTrace olay tetikleyicisi içerisinde TraceInfo.Message'daki log içeriğini kaydedebiliriz.

```
function TServerContainer1.DSServer1Trace(TraceInfo: TDBXTraceInfo):
CBRType;
begin
        LogInfo('Trace ' + TraceInfo.CustomCategory);
        LogInfo('   ' + TraceInfo.Message);
        Result := cbrUSEDEF; // take default action
end;
```

Unutmamak gerekir ki, TSQLConnection bileşenine bağlanan bir TSQLMonitor bileşeni ile istemci tarafta DataSnap istemcisi ve sunucusu arasındaki iletişimi takip etmek (trace) mümkündür. (Bu DataSnap Server için bu konu ile ilgili bir örnek yapacağım)

Örnek bir izleme çıktısı şu şekilde olabilir.

```
17:05:55.492 Trace
17:05:55.496    read 136 bytes:{"method":"reader_close","params":[1,0]}

{"method":"prepare","params":[-1,false,"DataSnap.ServerMethod",
                "TServerMethods1.AS_GetRecords"]}
17:05:55.499 Prepare
```

Görüldüğü gibi TraceInfo.Message çağrılan metodun byte sayısı ile ilgili bilgiyi içerir.

--- TDSServerClass ---

Bu bileşen, sunucu taraftaki sınıfı, uzak istemcilere yayınlanan (published) metodları sergilemek için özelleştirmeden sorumludur. (Dinamik metod yürütme kullanarak)

TDSServerClass bileşeni, TDSServer bileşenin işaret eden bir Server (sunucu) özelliğine sahiptir. Name ve Tag dışındaki diğer önemli özelliği ise LifeCycle özelliğidir. Bu özellik varsayılan olarak Session olarak atanır, ama Server veya İnvocation olarak da atayabiliriz. Server, sunucunun bütün ömrü için kullanılan bir sınıf varlığı, Session, DataSnap oturumunun bütün ömrü için kullanılan bir sınıf varlığı ve İnvocation, bir metodun yürütülmesi için kullanılan bir sınıf varlığı anlamına gelir. Session, her gelen bağlantının kendi sınıf arlığını oluşturacağı anlamına gelir. Eğer bunu İnvocation'a çevirirsek, durumsuz bir sunucu sınıfı ile sonlanmalıdır. (Eğer CGI web server uygulaması oluşturulacaksa gayet faydalıdır, çünkü durumsuzdur ve her istek için yükleme/boşaltma (load/unload) yapar). LifeCycle özelliğini Server olarak değiştirdiğimiz zaman tek bir sunucu sınıfı varlığının gelen bütün bağlantılar ve istekler için paylaşılacağını belirtir. Bu durum, eğer gelen bağlantıların sayısını tutmak gibi bir durum söz konusu ise gayet yararlıdır ama thread gerektirecek bir durumun olmaması gerekmektedir. (Çoklu isteklerin gelip aynı anda işlem görmek istemesi gibi durumlar)

TDSServerClass bileşeni, 4 adet olay barındırır. Bunlar; OnCreateInstance, OnDestroyInstance (bu, varlığın kesinlikle oluşturulduğu ya da yok edildiği durumlarda kullanılmalıdır), OnGetClass ve OnPrepare olaylarıdır. OnPreapre olayı, bir sunucu metodu hazırlanırken kullanılabilir. Delphi 2009'u kullanırken ya da Delphi 2010'u kullanıp manuel olarak TDSServerClass bileşenin eklediğimiz zaman OnGetClass olayını kendimizin elle belirtmesi gerekiyordu, hangi sınıfın sunucudan istemciye uzaktan yönetim için belirleneceğini belirtmek için. Ama Delphi 2010'daki DataSnap sihirbazı ile otomatik olarak OnGetClass olay tetiklenmesi kod kesimine eklenir.

```
procedure TServerContainer1.DSServerClass1GetClass(
DSServerClass: TDSServerClass; var PersistentClass: TPersistentClass);
begin
        PersistentClass := ServerMethodsUnitDemo.TServerMethods1;
end;
```

Bu otomatik türetilmiş bir kod olduğu için daha önceden otomatik oluşturulmuş unit olan ServerMethodsUnit1.pas'ın adını ServerMethodsUnitDemo olarak değiştirip buna göre gerekli düzenlemeleri yapmalıyız.

--- TDSTCPServerTransport ---

Bu bileşen, DataSnap sunucusu ve istemcisi arasında TCP/IP'yi iletişim protokolü olarak kullanarak gerekli iletişimi kurmakla görevli bileşendir.

Bu bileşenin BufferKBSize, Filters (Delphi 2010'la gelen bir özellik), MaxThreads, PoolSize, Port ve Server olmak üzere 6 adet özelliği bulunmaktadır. BufferKBSize özelliği iletişim için gerekli olan buffer boyutunu belirler ve varsayılan olarak 32 KB değerini alır. Filter özelliği 4. bölümde detaylı olarak anlatılacağı üzere, iletişim için filtre kolleksiyonu içeren bir özelliktir. MaxThread özelliği maksimum thread sayısı tanımlaması içindir (Limit olması istenilmiyorsa 0 olarak atanır). PoolSize, bağlantıları havuzlamak için, Port ise sunucunun istemciye bağlanması için gerekli olan TCP/IP portunu kontrol etmek için kullanılır. Server özelliği TDSServer bileşenin gösterir. TDSTCPServerTransport bileşeni ile ilgili olarak herhangi bir olay bulunmamaktadır.

--- TDSHTTPService ---

Bu bileşen, DataSnap sunucusu ve istemcisi arasındaki iletişimi HTTP iletişim protokolünü kullanarak kurmakla yükümlü olan bileşendir.

Bu bileşen Tag ve Name özellikleri dışında 10 adet özelliğe sahiptir. Bunlar; Active, AuthenticationManager, DSHostName, DSPort, Filters, HttpPort, Name, RESTContext, Server ve salt okunur (read-only) olan ServerSoftware özellikleridir.

Active özelliği DSHTTPService'in gelen talepleri dinleyip dinlemediğini kontrol edecek özelliktir. Bu tasarım zamanında (design time) True olarak atanabilir ama bu da DataSnap sunucusunun çalışma zamanında (run time) başlatılmasını engeller. (Çünkü aynı portu dinleyecek tek bir aktif DSHTTPService bileşeni olacak, ve çalışma zamanında başka bir tane başlatılamayacak). Bir TDSHTTPService başlatmanın en iyi yolu TServerContainer bileşeninin OnCreate olay tetikleyicisinde TDSHTTPService'in Active özelliğini True olarak atamaktır.

```
procedure TServerContainer1.DataModuleCreate(Sender: TObject);
begin
        DSHTTPService1.Active := True;
end;
```

AuthenticationManager özelliği HTTP yetkilendirmesi ile ilgili olarak ilgili bileşenin atanması ile alakalı özelliktir. Bizim örneğimizde bu özellik TDSHTTPServiceAuthenticationManager bileşenini gösterir ki zaten bu bileşene bir sonraki bölümde değineceğiz.

DSHostName ve DSPort özellikleri bağlanılacak DataSnap sunucusu ile ilgili bilgiler için kullanılır ama Server özelliğine bir değer atanmadıysa kullanılır. Çoğu durumda TDSServer bileşeninin Server özelliği kullanılır bu iş için.

Filter özelliği 4. bölümde detaylı olarak anlatılacağı üzere, iletişim için filtre kolleksiyonu içeren bir özelliktir.

HttpPort özelliği, DSHTTPService bileşenin gelen bağlantılar için dinleyeceği port bilgisini taşıyan özelliktir. Unutmamak gerekir ki bu özelliğin varsayılan değeri 80'dir ve eğer web servis (IIS gibi) kurulu olan bir makinada çalışıyorsanız, web servis de 80 portunu kullandığı için bağlantı port numarasını değiştirmeniz gerekir DataSnap için. Bunu yapmazsanız açıklayıcı olmayan karmaşık bir hata mesajı alırsınız uygulamayı çalıştırdığınız zaman.

RESTContext özelliği, DataSnap sunucusunu bir REST servisi olarak çağırabilmek için gerekli olan REST içeriğinin URL bilgisini taşır. Varsayılan olarak "rest" değeri atanır ki, biz sunucuyu " http://localhost/datasnap/rest/.." şeklinde çağırabilelim. Bunu 6. bölümde REST,JSON ve istemci geri dönüş metodlarında göstereceğim.

Son olarak Server özelliği, container (taşıyıcı) üzerinde var olan bir TDSServer bileşenine işaret eder. Eğer bu özellikle TDSServer'a bağlanmak istemiyorsak DataSnap sunucusuna bağlanmak için TCP protokolünü kullanarak DSHostName ve DSPort özelliklerini kullanabiliriz. Server özelliğine bir değer atandığı zaman, bu özelliklere atanan değerler görmezden gelinir.

TDSHTTPService bileşeni 5 olay içerir. Bunların 4ü REST ile ilişkili, bir tanesi de Trace ile ilişkilidir. REST ile ilişkili olanlara 6. bölümde detaylıca bakacağız.

OnTrace olayı, servise gelen taleplerin izinin tutulduğu olaydır. Şu şekilde gösterilebilir.

```
procedure TServerContainer1.DSHTTPService1Trace(Sender: TObject;
AContext: TDSHTTPContext; ARequest: TDSHTTPRequest;
AResponse: TDSHTTPResponse);
begin
        LogInfo('HTTP Trace ' + AContext.ToString);
```

```
        LogInfo('   ' + ARequest.Document);
        LogInfo('   ' + AResponse.ResponseText);
end;
```

Unutmamak gerekir ki HTTP İzleme (Trace) bilgisi, istemci taraf, sunucuya başlanmak için HTTP protokolünü kullandığı zaman çalışır. TCP/IP'de değil.

Çıkan örnek bir sonuç şu şekilde olur.

```
17:05:55.398 HTTP Trace TDSHTTPContextIndy
17:05:55.400    /datasnap/tunnel
17:05:55.403    OK
```

Bu demektir ki AContext'e TDSHTTPContextIndy, ARequest'e /datasnap/tunnel ve AResponse'e OK değerleri atanmıştır.

--- TDSHTTPServiceAuthenticationManager ---

Bu bileşen Server Container'a en baştaki diyalogda iletişim protokollerinde HTTP seçildiği zaman eklenir. Tabi bunu manuel olarak da eklememiz mümkün. Bu bileşen TDSHTTPService bileşeninin AuthenticationManager özelliğine bağlanmalıdır.

Bu bileşenin tek bir olayı vardır. Bu olay, OnHTTPAuthenticate olayıdır ve DataSnap istemcisinin sunucuya yaptığı bağlantı talebindeki HTTP yetkilendirmesini doğrulamak için kullanılır.

```
procedure
TServerContainer1.DSHTTPServiceAuthenticationManager1HTTPAuthenticate(
        Sender: TObject; const Protocol, Context, User, Password: string;
        var valid: Boolean);
        begin
                if (User = 'Bob') and (Password = 'Swart') then
                        valid := True
                else
                        valid := False
        end;
```

Kesinlikle yukarıda yazılan doğrulama ile ilgili kod parçasını, veritabanı içerisindeki kriptolanmış (hash) bir şifreyi kontrol etmek gibi örneklerle değiştirip gelitirebilirsiniz. Kullanıcı adı ve kriptolanmış şifre alanlarının istemciden sunucuya gönderildiği HTTP yetkilendirmesi için HTTPS kullanmak en iyi yoldur. Ve bu yüzden umuyorum ki Embarcadero ilerde diyalogdaki TCP/IP ve HTTP alanlarının yanına HTTPS alanını da ekleyecektir.

HTTPS, bağlantının güvenli ve bağlantı için gönderilen verinin kriptolanmış olduğundan emin olur, böylece dışarıdan biri sizinvar olan bağlantınıza ellerini uzatıp sizin kullanıcı adı ve şifre bilgilerinizi almaya teşebbüs edemez. Bu yüzden web sorumlunuz veya ISP'nizle görüşüp domaininizi HTTPS'e geçirmek için ne yapmanız gerektiği konusunda bilgisine danışın. Şiddetle tavsiye ediyorum ve örnek olarak kullandığım https://www.bobswart.nl adresini verebilirim.

Gerçek zamanlı (günlük hayatta) DataSnap sunucu uygulamaları için gayet güzel bir çözüm de HTTP yetkilendirme (teşebbüsler) işlemlerini Protokol ve Context bilgilerinin de yazıldığı bir log dosyasına kaydetmektir. Bu da size kimin bağlanıp, kimin bağlanmaya çalıştığına ilişkin bilgi verir. (Sizin DataSnap sunucunuza erişmek isteyen insanların yaptığı bağlantı girişimleri gibi..)

--- ServerMethodsUnitDemo ---

ServerContainerUnitDemo.pas gibi hayati derecede önemli bir dosyayı bitirdikten sonra şimdi de ServerMethodsUnitDemo.pas dosyasına geçebiliriz. Diyalogda biz TDSServerModule sınıfını ata sınıf olarak kullandık ve bu yüzden TServerMethods1 sınıfı da bizim TDSServerModule sınıfımızdan türedi (TDSServerModule TDSServerModuleBase sınıfından, o da TProviderDataModule sınıfından bir Destroy yıkıcısı (destructor) ve BeforeDestruction prosedürünü alarak türedi. TProviderDataModule, TDataModule sınıfından sağlayıcılarla çalışma özelliği eklenerek türemiştir ama bundan daha sonra çok bahsedeceğiz.)

TServerMethods1'in atalarında biri de TDataModule olduğu için tasarım kısmını gösterebilir ve tıpkı veri erişimi kontrolleri gibi görsel olmayan kontrolleri de buraya ekleyebiliriz. Veri erişimi bileşenlerini 3. bölümde ekleyeceğimiz için şimdilik buranın tasarım kısmını boş bırakıyoruz ve TServerMethods1 sınıfına ilgili metodları yapıştırıyoruz.

Eğer bu proje grubuna herhangi bir DataSnap Konsol Uygulaması veya Servis Uygulaması eklemek istemiyorsanız, bundan sonrası için 2.1.4. bölüme kadar atlayıp Sunucu Metodlarına direk geçiş yapabilirsiniz.

--- Çok Hedefli Proje Grubu - Konsol ---

Proje grubuna geri dönüp yeni bir proje ekleyebiliriz. Bu da bir konsol uygulaması olur. Proje grubu'na sağ tıklayıp Yeni Proje Ekle dedikten sonra daha önce karşımıza çıkan diyalogdan bu sefer Konsol Uygulaması kısmını seçeriz.

Burada önemli olan nokta ise, Konsol Uygulamasını seçtikten sonra diğer noktaların seçiminin önemli olmaması. Çünkü her halükarda ServerContainerUnitDemo ve ServerMethodsUnitDemo adlı unit dosyalarını kullanacağız.

Yeni projeyi oluşturduğu zaman varsayılan olarak bir proje dosyası (Sroject1.dproj) ve 2 adet unit dosyası (ServerContainerUnit2.pas, ServerMethodsUnit2.pas) oluşturuyor. Burada ServerMethodsUnit2.pas dosyasına sağ tıklayarak projeden sileriz. Diğer .pas dosyasının kalması lazım, çünkü onun içerisine bir metod kopyalayacağız. Projeyi DataSnapConsoleServer.dproj olarak kaydederiz. Yeni oluşturulmuş ServerContainerUnit2.pas ve daha önceden yazdığımız ServerContainerUnitDemo.pas dosyalarına baktığımız zaman aradaki farka baktığımızda ServerContainerUnit2.pas içerisinde global olarak tanımlanmış RunDSServer adında bir prosedür görürüz. Bu prosedür sadece konsol uygulamalarında işe yaradığı için bu kod parçasını kopyalar ve DataSnapConsoleServer.dproj dosyasında main bloğunun üstüne yapıştırırız.

Bu projenin şu anki halini derlerken oluşacak hataları proje dosyasının uses kısmına Windows ekleyerek çözmemiz mümkün. Şu anda DataSnapConsoleServer.dproj dosyası şu şekilde görünmekte.

```
program DataSnapConsoleServer;

{$APPTYPE CONSOLE}

uses
        SysUtils,
        Windows,
        ServerContainerUnit2 in 'ServerContainerUnit2.pas'
```

```
        {ServerContainer2: TDataModule};

    procedure RunDSServer;
    var
        LModule: TServerContainer2;
        LInputRecord: TInputRecord;
        LEvent: DWord;
        LHandle: THandle;
    begin
        Writeln(Format('Starting %s', [TServerContainer2.ClassName]));
        LModule := TServerContainer2.Create(nil);
        try
            LModule.DSServer1.Start;
            try
                Writeln('Press ESC to stop the server');
                LHandle := GetStdHandle(STD_INPUT_HANDLE);
                while True do
                begin
                    Win32Check(ReadConsoleInput(LHandle,
LInputRecord, 1, LEvent));
                    if (LInputRecord.EventType = KEY_EVENT)
and
                    LInputRecord.Event.KeyEvent.bKeyDown
and

    (LInputRecord.Event.KeyEvent.wVirtualKeyCode = VK_ESCAPE) then
                    break;
                end;
            finally
                LModule.DSServer1.Stop;
            end;
        finally
            LModule.Free;
        end;
    end;


    begin
        try
            RunDSServer;
        except
            on E: Exception do
            Writeln(E.ClassName, ': ', E.Message);
        end
    end.
```

Malesef proje şu anda 3 tane daha hata vermekte. Proje dosyası hala ServerContainerUnit2.pas dosyasını kullandığı için bu dosyayı sağ tıklamayla kaldırarak ve bu proje dosyasına sağ tıklayıp daha önceden oluşturduğumuz ServerContainerUnitDemo.pas dosyasını bu proje kapsamına da ekleyerek sorunu hallederiz.

Sonuç olarak içinde TServerContainer2 referansının geçtiği her yerde hata mesajı alacağız. ServerMethodsUnitDemo içerisinde TServerContainer1 tanımlanmış

olmalı, bu sebeple bu sıkıntıları gidermek için DataSnapConsoleServer proje dosyasının kodu içerisindeki TServerContainer2 geçen yerleri TServerContainer1 olarak değiştirmemiz gerekir. (Toplamda 3 yerde var)

Bu işlemleri gerçekleştirdiğimiz zaman DataSnapConsoleServer.dproj projesini de tıpkı orjinal DataSnapServer.dproj projesi gibi derlememiz mümkün olacaktır. 2 hedef projede de ServerMethodsUnitDemo.pas gibi ServerContainerUnitDemo.pas dosyası da paylaşılabilir.

--- Çok Hedefli Proje Grubu - Windows Servisi ---

Ve daha önce oluşturduğumuz projeye eklenecek tek bir hedef kaldı, o da Windows Servis Uygulaması. Gene aynı şekilde var olan projeye yeni proje ekleme kısmından servis uygulamasını seçiyoruz ve bütün iletişim protokollerini de seçiyoruz. Birazdan görebileceğimiz gibi Servis uygulaması için Service Container VCL Form veya Konsol için olan Server Container'den biraz farklı olsa da bütün iletişim protokollerini seçtiğimizden emin olmalıyız. Haliyle yeni eklenen projeden dolayı bir proje dosyası ve 2 adet .pas dosyası da eklenecektir proje grubuna (ServerContainerUnit, ServerMethodsUnit). ServerMethodsUnit dosyasını daha önce de yaptığımız gibi servis projesinden kaldırıp onun yerine daha önceden oluşturduğumuz ServerMethodsUnitDemo.pas dosyasını ekleriz.

Ama yeni oluşturulan ServerContainerUnit1.pas diğerlerinden biraz daha farklıdır. TDSServer, TDSServerClass ve iletişim bileşenlerini kullanan bir TDataModule yerine DataSnap bileşenlerini içeren TService sınıfından türemiş bir sınıfımız bulunmaktadır. TService tarafından türemiş olsa da ek olarak üzerinde 4 adet metod bulunmaktadır. Bunlar; servisin Stop, Pause, Continue ve Interrogate olaylarıdır.

```
type
TServerContainer3 = class(TService)
        DSServer1: TDSServer;
        DSTCPServerTransport1: TDSTCPServerTransport;
        DSHTTPService1: TDSHTTPService;
        DSHTTPServiceAuthenticationManager1:
TDSHTTPServiceAuthenticationManager;
        DSServerClass1: TDSServerClass;
        procedure DSServerClass1GetClass(DSServerClass: TDSServerClass;
        var PersistentClass: TPersistentClass);
        procedure ServiceStart(Sender: TService; var Started: Boolean);
        private
                { Private declarations }
        protected
                function DoStop: Boolean; override;
                function DoPause: Boolean; override;
                function DoContinue: Boolean; override;
                procedure DoInterrogate; override;
        public
                function GetServiceController: TServiceController; override;
end;
```

Diğer bir deyişle Konsol uygulamasını eklerken yaptığımız gibi ServerContainerUnitDemo dosyasını bu Servis uygulaması için de paylaşamayız. Ayrıca servis projesinin Container dosyasının adını da ServerContainerUnitServiceDemo.pas

olarak değiştirmeliyiz. Haliyle proje dosyasının da adını DataSnapServiceServer.dproj olarak değiştirip kaydetmeliyiz.

ServerContainerUnitServiceDemo.pas dosyası içerisindeki ServerMethodsUnit1.pas dosyasının referanslarını ServerMethodsUnitDemo.pas olarak değiştirmeliyiz. Ve bundan sonra bütün bu 3 hedef (Form, Konsol, Servis projeleri) tarafından paylaşılacak Sunucu Metodlarını (Server Methods) yazmaya başlayabiliriz.

--- Sunucu Metodları ---

Eğer bir veya daha fazla DataSnap sunucu projemiz varsa ve bunların tamamı ServerMethodsUnitDemo unit'ini kullanıyorsa bu durumda sunucu metodlarını detaylı olarak oluşturmamız ve incelememiz gerekmektedir.

Daha önce de bahsettiğim gibi TServerMethods1 sınıfı DataSnap istemcilerine gösterilecek metodları gösterecek bir DataSnap sunucu nesnesidir. Eğer "Örnek metodu dahil et" (İnclude sample method) seçeneğini işaretlersek bu sınıfın içerisinde zaten örnek bir metod otomatik olarak oluşturulacaktır. (EchoString fonksiyonu). Hadi DataSnap sunucu makinasındaki zamanı döndüren yeni bir metod ekleyelim. Bunu yapabilmek için sınıfın tanımında 2 fonksiyonu da tanımlamak gerekir. Örnek kod aşağıdaki gibidir.

```
type
TServerMethods1 = class(TDSServerModule)
        private
                { Private declarations }
        public
                { Public declarations }
                function EchoString(Value: string): string;
                function ServerTime: TDateTime;
end;
```

ServerTime metodunu oluşturmak EchoString metodundaki gibi son derece kolaydır.

```
function TServerMethods1.EchoString(Value: string): string;
begin
        Result := Value;
end;

function TServerMethods1.ServerTime: TDateTime;
begin
        Result := Now
end;
```

Sunucu uygulamasını şimdi derleyip çalıştırabiliriz. Eğer birden çok hedef proje oluşturduysanız bunu test etmenin en iyi ve kolay yolu DataSnapServer uygulamasıdır. Windows versiyonunuz ve güvenlik seviyenize göre Windows Firewall'un DataSnap uygulamanızın bazı özelliklerini kısıtladığına dair bir güvenlik uyarısı gelebilir.

Bu durumi DataSnap uygulamanızın TCP/IP ve HTTP'den gelen istekleri dinlemesinden kaynaklanan bir durumdur. Bir Truva Atı gibi gelen taleplerin tamamını dinlemek istiyorsak çıkan uyarıda "erişime izin ver" diyerek uygulamayı çalıştırmaya devam etmeliyiz.

--- DataSnap ıstemcisi ---

İlk DataSnap sunucu demo uygulamamızı yazıp çalıştırdıktan ve gelen talepleri dinlemeye başladıktan sonra artık bir DataSnap istemcisi oluşturmaya başlayabiliriz. Bu bölümde istemci taraftan sunucuya nasıl bağlanılacağını ve oluşturulmuş sunucu sınıflarındaki metodları istemciye nasıl dahil edebileceğimizi göstereceğiz.

Öncelikle DataSnap sunucusunun çalıştığından emin olup, sonrasında istemci uygulama projesini oluşturmalıyız. DataSnap Sunucu projeleri ile DataSnap istemci projeleri arasındakki geçişleri tasarım zamanında (design time) kolayca yapabildiğimiz için, DataSnap İstemci projesini de daha önce oluşturduğumuz proje grubunun içine oluştururuz. Herhangi bir proje DataSnap istemci projesi olabilir ama bu demo için ben VCL Form uygulaması geliştireceğim ve oluşan projeyi DataSnapClient.dpr ve ClientForm.pas olarak kaydedeceğim. Bileşen Listesi (Tool Pallette) içerisinde DataSnap Sunucu'su ile ilgili olarak 6 adet bileşen bulunmasına rağmen, DataSnap İstemci ile ilgili şu anda kullanacağımız herhangi bir bileşen bulunmamaktadır. (Normalde var ama)

DataSnap İstemci ile ilgili olarak bulunan bileşenler, hala kullanılabilecek eski DataSnap bileşenleridir, ama yeni DataSnap mimarisini kullanmak için tavsiye edilen bir yöntem değildir bu bileşenleri kullanmak. Bunun tek istisnası, eski bir DataSnap sunucusunun yeni bir DataSnap istemcisine bağlanması için kullanılan yeni TDSProviderConnection bileşenidir. (Bunu 3.2 bölümünde anlatacağım)

DataSnap İstemci bölümündeki bileşenler yerine dbExpress bölümündeki bileşenleri kullanmak daha iyi olur. Mesela TSQLServerMethod gibi bir bileşen kullanılabilir. (Bir sonraki ekran görüntüsünde görüleceği üzere bu bileşeni tanımlamak oldukça kolaydır, şu andaki dbExpress bileşenlerini sahip olduğu TSQL yerine kendi TSQL'ine sahiptir.)

Bu bileşen DataSnap sunucusu içerisindeki metodları uzaktan çağırmak için kullanılır, ama bu işlemi gerçekleştirebilmesi için öncelikle sunucuya bağlanması gerekir.

Bu bağlantı daha önceki TxxxConnection bileşenleri kullanılmayıp, TSQLConnection bileşeni kullanarak yapılabilir. Bu işlemi kolaylaştırmak için bileşenin içindeki Sürücü (Driver) listesi içerisinde DataSnap de bulunmaktadır.

Bu yüzden ClientForm dediğimiz istemci proje formu üzerine bir adet TSQLConnection bileşeni yerleştirip, Sürücü özelliğini DataSnap olarak atarız. Bu işlem sonunda Sürücü özelliği artı işareti alır ve böylece Sürücü özelliği altındaki bütün alt özellikleri görebiliriz.

Unutmamak gerekir ki, eğer CommunicationProtocol özelliği eğer boş bırakılmış ise, TSQLConnection bileşeni bu özellikte varsayılan değer olarak TCP/IP'yi kullanır. (port 211) BufferKBSize özelliği varsayılan olarak 32 KB, Port değeri 211 olarak (Sadece sunucu tarafında) atanır. Gerçek zamanlı uygulamalarda ben mutlaka Port numarasını 211 dışında bir değer olarak atarım (hem sunucu hem istemci tarafında), çünkü DataSnap sunucu uygulamalarında TCP/IP için varsayılan port numarasının 211 olduğu oldukça yaygın olarak bilinen bir gerçektir.

HostName, UserName ve Password özellikleri DataSnap sunucusuna bağlanmak için kullanılan özelliklerdir. Lokal alanda yapılan bir test için HostName alanına "localhost" yazılır, ama genelde bu özelliğin değeri için herhangi bir host adı, DNS adı veya direkt IP numarası girer.

Unutmamak gerekir ki TSQLConnection bileşeninin LoginName özelliğini bağlantı diyalog penceresi açılmaması için False olarak atamak lazım.

TSQLConneciton bileşeninin Driver (Sürücü) özelliğinin ilgili değerlerini girdikten sonra Bağlandı (Connected) özelliğini True olarak atadığımız zaman DataSnap

Sunucusuna bağlantı kurmaya çalışır. Ama daha önce de belirttiğim gibi bu işlemler sırasında DataSnap sunucusunun açık olmasına dikkat edilmelidir.

--- DataSnap ıstemci Sınıfları ---

Bağlantının yapıldığını doğruladıktan sonra TSQLConnection bileşenin sağ tıklayıp DataSnap İstemci Sınıflarını Oluştur (Generate DataSnap Client Classes) seçeneğini seçtiğimiz zaman TServerMethods1Client adında bir sınıf içeren ve varsayılan olarak Unit1.pas olarak adlandırılan bir dosya oluşturur. Bu dosyayı ServerMethodsClient.pas olarak kaydetmeliyiz. (Sunucu taraftaki DataSnap Sunucu metodlarının bulunduğu sınıf isminin sonuna "Client" kelimesi getirilerek oluşturulur varsayılan olarak IDE tarafından)

Kaydetmeden önceki haliyle oluşturulan sınıfın kaynak kodu aşağıdaki gibidir.

```
type
TServerMethods1Client = class
        private
                FDBXConnection: TDBXConnection;
                FInstanceOwner: Boolean;
                FEchoStringCommand: TDBXCommand;
                FServerTimeCommand: TDBXCommand;
        public
                constructor Create(ADBXConnection: TDBXConnection);
overload;
                constructor Create(ADBXConnection: TDBXConnection;
                AInstanceOwner: Boolean); overload;
                destructor Destroy; override;
                function EchoString(Value: string): string;
                function ServerTime: TDateTime;
end;
```

Burada da gördüğümüz gibi sınıfın içerisinde 2 adet yapıcı (Constructor), 1 adet yıkıcı (Destructor), ve DataSnap sunucu tarafında oluşturduğumuz 2 adet donksiyon bulunmaktadır (EchoString ve ServerTime).

Bu metodları kullanabilmek için ClientForm'un uses kısmına ServerMethodsClient'i eklememiz gerekir. Sonrasında form üzerine bit buton yerleştirerek ve OnClick olay tetiklemesinin içerisine aşağıdaki kaynak kodu yazarak ilgili fonksiyonları kullanabiliriz.

```
procedure TForm2.Button1Click(Sender: TObject);
var
        Server: TServerMethods1Client;
begin
        Server :=
TServerMethods1Client.Create(SQLConnection1.DBXConnection);
        try
                ShowMessage(DateTimeToStr(Server.ServerTime))
        finally
                Server.Free
        end;
end;
```

Bu kaynak kod, TServerMethods1Client türünden bir varlık oluşturur, ServerTime sunucu metodunu çağırır ve en sonunda DataSnap sunucusuna olan proxy'yi yok eder.

Butona tıklandığı zaman sonuç olarak beklendiği gibi sunucunun o anki zaman bilgisi kullanıcıya gösterilir.

Bu örneği okuyucu olan sizlerin benzer bir yöntemle EchoString metodunu denemesi için burada sonlandırıyorum.

--- HTTP ıletişim Protokolü ---

Daha önceden TSQLConnection bileşenindeki Driver özelliğinin ConnectionProtocol alt özelliğinin varsayılan olarak TCP/IP olarak geldiğini belirtmiştim. Bu da demektir ki herhangi bir HTTP İzleme mesajını göremeyeceğiz. Buna rağmen bu durumu değiştirmek hiç de zor değildir. Sadece ConnectionProtocol özelliğini HTTP olarak değiştirmemiz yeterlidir. Yapılan bu değişiklik yüzünden Port özelliğini de değiştirmek gerekir. (Çünkü TCP/IP için olan 211 görünüyor) Bu yüzden ServerContainer içerisindeki TDSHTTPService içerisindeki Port değerine özelleştirilmiş bir değer vermemiz gerekir.

Bu değişiklikleri yaptıktan sonra DataSnap İstemci uygulamasını çalıştırdığımız zaman bir uygulama hatası ile karşılaşırız.

Bu hatayı gidermek için DataSnap İstemcisindeki ClientForm'un uses kısmına DSHTTPLayer değerini eklememiz gerekir.

--- HTTP Yetkilendirme ---

HTTP protokolünü kullanmanın yararlarından biri de HTTP yetkilendirmesini de dahil edebilme yeteneğidir. Bu durum DataSnap Sunucusu tarafında TDSHTTPAuthenticationManager bileşeni tarafından desteklenmektedir.

OnHTTPAuthenticate olay tetikleyicisi tetiklendiği zaman burada HTTP Yetkilendirmesi ile ilgili bir kontrol yapılır ve TDSHTTPAuthenticationManager bileşeninin bize erişim için izin vermesi için doğru bilgileri gönderdiğimizden emin olmalıyız. Bunu yapmadığımız zaman yetkilendirmenin bulunmadığına dair bir hata mesajı alırız.

İstemciden sunucuya yapılan HTTP yetkilendirmesi işleminden başarıyla geçebilmemiz için TSQLConnection içerisindeki DSAuthUser ve DSAuthPassword değerlerini ClientForm içerisinde doldurmamız gerekmektedir.

Unutmamak gerekir ki eğer istemci ve sunucu uygulamaları aynı lokal makina üzerinde çalıştırmıyorsak bu bileşen üzerindeki HostName alanına özel bir değer yazmamız gerekmektedir.

--- DataSnap Sunucu Dağıtımı ---

Bu örnek, sunucu ve istemci uygulamaları aynı lokal makinada çalıştırıldığı için gayet güzel bir şekilde çalışır ama gerçek hayatta karşılaşacağımız durumlarda DataSnap sunucusu bir sunucu makinada, bir veya daha fazla istemci ise ağ üzerinde sunucu makinasına bağlanacak şekilde çalışacaktır. Sunucu uygulamanın dağıtılıp yayınlanacağı makina genelde üzerinde Delphi kurulmamış bir makina olacaktır. Bu da demektir ki çalışma zamanlı paketler olmadan DataSnap Sunucu uygulamasını derlemek ortaya büyük boyutlu bir uygulama dosyası çıkar. Veri erişimi bileşenleri (Data Access Components) kullanmadığımız sürece herhangi ek bir veritabanı sürücüsüne veya ekstradan DLL dosyasına ihtiyacımız bulunmamaktadır.

--- DataSnap ıstemci Dağıtımı ---

DataSnap İstemci uygulamasının DataSnap Sunucu uygulamasından farklı bir makinada çalıştıracağı gerçeğini göz önünde bulunduracak olursak, istemcinin sunucuya bağlanabildiğinden emin olmamız gerekmektedir. Bunu gerçekleştirmek için TSQLConnection bileşeninin içerisinde sadece Driver özelliğinin alt özellikleri olan ConnectionProtocol ve Port özellikleri değil, aynı zamanda HostName özelliği de doldurulmalıdır. Belirli bir IP adresi yazılmamalı, ama eğer mümkünse mantıksal DNS adı kullanılmalıdır. Benim kendi DataSnap sunucularım için mesela, ben HostName özelliğine "www.bobswart.nl" değerini vermekteyim. (Eğer CommunicationProtocol özelliğinde aktüel protokol bilgisi belirtilmişse HostName değerini girerken "http://" değerini eklememize gerek yoktur.)

--- DataSnap ve Veritabanı ---

Delphi 2010 DataSnap framework'ünü basit sunucu metodları için kullanabileceğimiz gibi, veritabanı erişimlerini de ekleyip, mimariyi çok katmanlı bir veritabanı uygulaması haline de getirebiliriz. (DataSnap sunucusu veritabanına bağlanır, ama DataSnap istemcileri thin veya smart client olarak, istemci tarafta her hangi bir veritabanı sürücüsüne (database driver) gerek kalmadan sunucuya bağlanabilir.)

Daha önce oluşturduğumuz DataSnap örneğinde sadece TSQLConnection bileşenini kullandık ama istersek ek olarak 2 yeni DataSnap bileşeni olan TsqlServerMethod ve TDSProviderConnection da kullanabiliriz.

Öncelikle sunucunun bir data seti (TDataSet) gösterdiğinden emin olmamız gerekir ve bunun için ServerMethodsUnitDemo dosyasına gelip data modülünün içerisine bir adet TSQLConnection bileşeni eklememiz gerekir. Bu bileşenle favori veritabanı ve tablomuza bağlanırız. (Ben burada BlackFish Employess veritabanındaki Employees tablosuna bağlanacağım) Bunu yapabilmek için ServerMethods1'in dizayn tarafına bir TSQLConnection bileşeni ekleyip Driver özelliğine BlackFish değerini atamalıyız. Sonra Driver özelliğinde Database (Veritabanı) özelliğine employee.jds dosyasının yolunu (path) vermemiz gerekir. (Bu dosyanın yolunun ne olduğu daha önce anlatıldı) TSQLConnection bileşeninin LoginPrompt özelliğinin False olmasına dikkat etmemiz gerekir ve sonrasında veritabanına bağlanmak için Connected özelliğine True değerini atarız.

CommandType özelliğini ctQuery olarak atadıktan sonra CommandText özelliğinin içine girerek bir sorgu cümleciği (query) yazarız.

Tasarım zamanında TSQLConnection bileşeninin LoginPrompt ve Connected özelliklerinin değerlerinin False olmasına dikkat etmemiz gerekir. Aynı şekilde TSQLDataSet bileşeninin Active özelliğinin de False değerini alması gerekir.

Şimdi TSQLDataSet'in içerik bilgisini döndürecek bir public türdeki fonksiyonu ServerMethodsUnitDemo dosyasındaki TServerMethods1 sınıfının içerisine yazabiliriz.

```
type
TServerMethods1 = class(TDSServerModule)
        SQLConnection1: TSQLConnection;
        SQLDataSet1: TSQLDataSet;
        private
                { Private declarations }
        public
                { Public declarations }
                function EchoString(Value: string): string;
                function ServerTime: TDateTime;
```

```
                function GetEmployees: TDataSet;
    end;
```

Yukarıdaki kodda da görebildiğimiz gibi TServerMethods1 sınıfının tanımlamasının içerisinde yeni fonksiyonumuz GetEmployee olarak adlandırılmaktadır ve içeriği de aşağıdaki gibidir.

```
    function TServerMethods1.GetEmployees: TDataSet;
    begin
        SQLDataSet1.Open; // make sure data can be retrieved
        Result := SQLDataSet1
    end;
```

Sunucuyu tekrar derleyip çalıştığından emin olmamız gerekir. Unutmamak gerekir ki, DataSnap sunucusunu kapattıktan sonra hala projeyi derleyemiyorsak, bu DataSnap sunucu projesinin hala çalıştığını gösterir.

--- TSQLServerMethod ---

DataSnap istemci uygulamasına geri dönüyoruz. Buradaki TSQLConnection bileşeninin sunucuya bağlı olarak kalmaması gerekiyor. (Eğer sunucuya hala bağlıysa, bu sunucu projesini derleyemememizin bir sebebi olabilir, işlem hala çalıştığından) Artık bileşenin Connected özelliğini tekrar True olarak atayarak, sunucudan bilgileri güncelleyip tekrardan istemci sınıfı sağ tıklama metoduyla oluşturabiliriz. Daha önceki ServerMethodsClient dosyasının üzerine yazmaması için öncelikle eski dosyayı projeden silip sonrasında tekrardan sınıfı oluşturma işlemini gerçekleştiririz. Yeni oluşturulan dosyayı ServerMethodsClient.pas olarak kaydedersek, daha önceki istemci kodu değiştirmemize gerek kalmaz. İstemci sınıfı oluştur seçeneğini tekrardan çalıştırdığımız zaman, TServerMethods1Client sınıfı daha önce tanımladığımız GetEmployee fonksiyonunu da gösterecek şekilde daha önceki haline göre genişletilmiş bir şekilde karşımıza çıkar. Tekrardan oluşturduğumuz istemci sınıfın kaynak kodu şu şekilde olacaktır.

```
    type
    TServerMethods1Client = class
        private
            FDBXConnection: TDBXConnection;
            FInstanceOwner: Boolean;
            FEchoStringCommand: TDBXCommand;
            FServerTimeCommand: TDBXCommand;
            FGetEmployeesCommand: TDBXCommand;
        public
            constructor Create(ADBXConnection: TDBXConnection);
overload;
            constructor Create(ADBXConnection: TDBXConnection;
            AInstanceOwner: Boolean); overload;
            destructor Destroy; override;
            function EchoString(Value: string): string;
            function ServerTime: TDateTime;
            function GetEmployees: TDataSet;
    end;
```

TDataSet içerisindeki veriye erişmek için GetEmployee fonksiyonunu çalıştırmak için Tool Pallette kısmındaki dbExpress bölümünden bir TsqlServerMethod bileşeni kullanabiliriz. Bu bileşeni istemcinin formuna yerleştirip SQLConnection özelliğine SQLConnection1 değerini atayıp, ServerMethodName özelliğinden çağırabileceğimiz metodların listesine bakabiliriz. Bunlar; belirli sayıdaki DSAdmin metodları (TDSServer bileşenindeki HideDSAdmin özelliğine True değeri atılarak gösterilmeyebilir), 3 adet DSMetaData metodu, 7 adet TServerMethods.AS_xxx metodu (orjinal IAppServer arayüzü ile ilgili) ve en son olarak TServerMethods1 sınıfı içerisindeki EchoString, ServerTime ve GetEmployee metodlarıdır.

Şimdiki örneğimiz için GetEmployee metodunu ServerMethodName özelliğine değer olarak atamamız gerekmektedir.Bu değerle SqlServerMethod'un sonucu içerisinde işçi kayıtlarından oluşan bir data seti olacaktır. (Ya da yazdığımız sorgu cümleciğinden elde edilecek değer)

Artık, TDataSetProvider, TClientDataSet ve TDataSource gibi bileşenleri kullanarak örnekteki verinin TDBGrid üzerinde gösterilmesini sağlayabiliriz.

İstemcinin formuna bir adet TDataSetProvider bileşeni ekleyip DataSet özelliğine SqlServerMethod değerini atarız. Sonrasında aynı formun üzerine bir TClientDataSet bileşeni atarak ProviderName özelliğine DataSetProvider bileşenin değer olarak atarız. RemoteServer özelliğini boş bırakmalıyız, çünkü bu özellik eski DataSnap mimarisi için yapılmış olup, yenisinde kullanılmasına gerek yoktur.

En sonunda formun üzerine bir TDataSource bileşeni ekleyerek DataSet özelliğine TClientDataSet bileşenini değer olarak atarız. Artık TDBGrid ve TDBNavigator kullanarak, DataSource özelliklerine TDataSource bileşenini atayarak verileri istemci form üzerinde gösterme imkanına sahip oluruz.

Bağlantıyı tasarım zamanında test etmek için ClientDataSet bileşenin Active özelliğini True olarak atarız. (Bunu veriyi görebilmek için yapar, sonrasında özelliğe tekrar False değerini atarız) Çünkü bu yaptığımız TSQLConnection bileşeninin Connected özelliğini True ederek sunucu ve istemci arasındaki bağlatıyı oluşturmaya çalışır.

Bağlantı aktif olduğu zaman sonuç olarak TClientDataSet ve TSQLConnection bileşenleri de aktif olur ve veriyi TDBGrid üzerinden görmemiz mümkün olur.

Bu veriyi görüp, salt okunur olarak içine bakmak için en kolay yöntemdir. Unutmamak gerekir ki, TSQLServerMethod bileşeni TDataSetProvider ve TClientDataSet bileşeni çiftine istemci taraftan sunucu taraftaki veritabanına herhangi bir güncelleme için izin vermediğinden bir üstteki cümlede salt okunur ibaresini kullandım.

TSqlServerMethod bileşeni, salt okunur olarak veriye bağlanmak için az yük getiren ve uygun bir yöntemdir. Bu demektir ki DataSnap sunucusundan insanların veriler üzerinde modifikasyon yapamayacağı şekilde salt okunur olarak verileri göstermek istiyorsak, TServerMethods1 sınıfından TSQLDataSet bileşenindeki sonuçları göstermek iyi bir fikirdir.

Buna rağmen bazı durumlarda veriler üzerinde güncelleme gerekeceğinden veriyi göstermek için farklı bir bakış açısı izlemek gerekir.

--- TDSProviderConnection ---

Eğer güncellemeleri uygulamak istiyorsak sunucu taraftaki DataSetProvider'e referans olacak şekilde bir TDSProviderConnection bileşeni kullanmamız gerekir ki, bu sayede veriyi sadece okumayıp aynı zamanda güncellemeleri de gönderebiliriz.

Öncelikle sunucu taraftaki sunucu data modülünde birdeğişiklik yapmamız gerekir. TDataSet'i döndürecek bir fonksiyon yazmak dışında modüle bir TDSProviderConnection bileşeni ekleyerek bunun DataSnap sunucusundan DataSnap

istemcisine aktarılacağından emin olmamız gerekir. Bu yüzden ServerMethodsUnitDemo dosyasına geri dönerek TDataSetProvider bileşenini yerleştirerek, TSQLDataSet bileşenini de ekleyerek TDataSetProvider bileşeninin DataSet özelliğine TSQLDataSet bileşenini değer olarak atarız. Ayrıca TDataSetProvider bileşenini adını mantıklı bir hal alacak şekilde dspEmployees olarak değiştirmeliyiz.

Artık DataSnap sunucusunu tekrar derleyerek çalıştırabiliriz ve istemci tarafta verinin gösterilmesi ve güncellenmesi ile ilgili yapılacak değişikliklere başlayabiliriz.

--- TDSProviderConnection ıstemci Tarafı ---

Gösterilmiş TDataSetProvider bileşenini kullanabilmek için DataSnap istemcisi tarafında değişiklik yapmak için öncelikle istemci formdaki TSQLServerMethod ve TDataSetProvider bileşenlerini silmemiz, sonrasında bunların yerine bir TDSProviderConnection bileşeni eklememiz gerekmektedir. TDSProviderConnection bileşeninin SQLConnection özelliğine DataSnap sunucusuna bağlantı kuran TSQLConnection bileşenini değer olarak atamalıyız. Ayrıca TDSProviderConnection bileşeninin ServerClassName özelliğine de bir değer girmemiz gerekmektedir. Malesef bu özellik için bir sürükle bırak listesi (drop-down list) bulunmamaktadır. Şimdilik TDSServerModule sınıfının ismini elle girmeliyiz ki bu bizim örneğimizde TServerMethods1'dir.

Önceki örnekte TClientDataSet sadece ProviderName özelliği ile DataSetProvider1'e bağlanmıştı. Bununla beraber TDSProviderConnection bileşenini kullanırken öncelikle TClientDataSet bileşeninin RemoteServer özelliğini bu bileşene atamalı, sonra ProviderName özelliği için yeni bir değer ataması yapmalıyız. (Şu anda hala DataSetProvider1 olarak gözükmekte ama biz bunu dspEmployee olarak güncellemeliyiz ki bu DataSnap sunucusundan gösterilen TDataSetProvider bileşeninin isminden daha açıklayıcı bir isimdir.)

ProviderName özelliğini seçmek için kullanılacak sürükle bırak combobox içerisinde artık dspEmployee değeri görülmelidir. (Sunucu data modülü içerisinde yer alan TDataSetProvider bileşenini ismidir.)

Artık TClientDataSet bileşeni içerisindeki Active özelliğine True değerini atayarak, tasarım zamanında verileri görmemiz tekrardan mümkün olacaktır.

TClientDataSet bileşeninin Active özelliğine True değerini atamak, aynı zamanda TSQLConnection bileşeninin Connected özelliğini de otomatik olarak True atamayı sağlar. Unutmamak gerekir ki istemci form üzerinde bu iki özelliği de True olarak atamak iyi bir fikir değildir. En başta Delphi IDE'si içerisinde DataSnap istemci projesini açtıysak, DataSnap sunucusuna bağlantı kurmaya çalışacaktır, eğer server açık ve çalışmıyorsa hata verecektir bu bağlantı girişimi. İkincisi, uygulamayı çalışma zamanında başlattığımızda ve herhangi bir bağlantı olmadığında uygulama bir istisnai durum (exception) belirterek hata verecektir. Bu, uygulamayı lokal veri ile kullanmayı, bağlantı yoksa gereksiz bir şekilde vermeyi engeller.

Daha iyi bir yol, bir buton veya menü seçeneği ekleyerek TSQLConnection bileşenine dışarıdan erişerek TClientDataSet'i de dışarıdan aktif veya pasif hale getirmektir. Bu ileride anlatacağımız kullanıcı adı ve şifre bilgisini dahil etmek için de mükemmel bir andır. Şimdilik TClientDataSet bileşeninin Active özelliğine ve TSQLConnection bileşeninin de Connected özelliğine False değerini atamak yeterlidir. Sonra istemci forma bir buton ekleyerek OnClick olay tetikleyicisini TClientDataSet bileşenini dışarıdan açmak için kullanabiliriz.

```
procedure TForm2.Button2Click(Sender: TObject);
```

```
begin
        ClientDataSet1.Open;
end;
```

İstemci tarafta veriyi değiştirmek ve yapılan değişiklikleri sunucuya göndermek için birz zaman ayırıp kod eklememiz gerekir.

--- Veritabanı Güncellemeleri ---

İstemci tarafında yapılan değişikliklerin sunucu tarafına gönderilmesi için 2 farklı yöntem vardır. Otomatik veya elle. İkisi de en sonunda aynı metodu çağırır ama ikisinin de avantaj ve dezavantajları vardır.

Otomatik yaklaşımda veri üzerinde değişiklik yapmakla ilgili olarak TClientDataSet bileşeninin OnAfterInsert, OnAfterPost ve OnAfterDelete olay tetikleyicileri kullanılır. Olay tetikleyicilerinde, hepsi tek bir tanımlamada paylaşılabilir, değişiklikleri sunucuya göndermek için ApplyUpdates adında bir fonksiyoni ve veritabanında yapılan değişiklikleri geri çekmek için "Delta" çağrılır.

```
procedure TForm2.ClientDataSet1AfterPost(DataSet: TDataSet);
begin
        ClientDataSet1.ApplyUpdates(0);
end;
```

Eğer güncelleme sırasında kötü bir durum oluşursa, kaydın bulunmaması gibi, bir sonraki bölümde detaylı incelenecek olan TClientDataSet bileşenine ait OnReconcileError olay tetikleyicisi ile geri yükleme alınabilir.

Elle çözüm yönteminde gene aynı şekilde TClientDataSet'in ApplyUpdates fonksiyonu kullanılır ama OnAfterUpdate, OnAfterPost ve OnAfterDelete olay tetikleyicilerinde değil de mesela bit butonun OnClick olay tetikleyicisinde yapılır.

```
procedure TForm2.btnUpdateClick(Sender: TObject);
        ClientDataSet1.ApplyUpdates(0);
end;
```

Kullanıcının ApplyUpdates fonksiyonunu otomatik olarak kullanmasının avantajı, her hangi bir yerde yapılacak değişikliği unutmamasıdır. Ama bununla birlikte bir dezavantajı vardır ki, yapılan bir değişikliğin geri alınması gibi bir durum söz konusu değildir. Mesela veri post edildiği anda sunucuya aktarılır. Diğer taraftan elle yapıldığı zaman bu işlem, yapılan bütün değişiklikler, TClientDataSet bileşeninin belleğinde tutulur. Bu da kullanıcıya yapılan değişiklikleri geri alma şansı verir. Son değişikliği geri alma, özel bir kayıtta yapılan değişikliği geri alma, bütün güncelleme işlemlerini geri alma gibi. Güncelleme butonuna tıklandığı zaman kullanıcı hazır olduğunda ApplyUpdates fonksiyonunu gene çalıştırır. Buradaki potansiyel tehlike de kullanıcının güncelleme ile ilgili butona tıklamayı unutup yapılan değişikliklerin sadece TClientDataSet bileşenin belleğinde kalması ve bundan dolayı veri kaybı oluşma ihtimalidir. TClientDataSet içerisinde kalan değişikliklerin kontrolü gene aynı bileşen içerisindeki ChangeCount özelliğine bakarak kontrol edilebilir.

--- Hata Uzlaştırmak ---

TClientDataSet bileşeninin ApplyUpdates metodunun aldığı tek bir argüman vardır. O da güncellemeleri uygulamayı durdurmadan önce alınmasına izin verilecek

maksimum hata sayısıdır. İki istemcinin DataSnap sunucu uygulamasına bağlanarak, işçi kayıtlarına baktığını ve aynı kayıt üzerinde değişiklik yaptığını düşünelim. Geliştirdiğimiz yapıya göre ikisinin de değişikliği yapıp sonrasında ApplyUpdates metoduyla bu değişikliği sunucuya kaydedeceğini düşünelim. Eğer ApplyUpdates metodunun maksimum hata argümanını 0 olarak gönderirsek, bir taraf güncelleme işlemini başarıyla gerçekleştirirken, diğerinin güncelleme çalışması başarısız olacaktır. İkinci istemcinin güncellemeyi yapabilmesi için metodun argümanı olan maksimum hata sayısını 0'dan büyük bir değer olarak göndermesi gerekmektedir. Bununla birlikte eğer kullanıcı argüman olarak -1 değerini gönderirse daha önceki istemci tarafından güncelleme yapılmış hiç bir kayda güncelleme işlemini gerçekleştiremez. Diğer bir deyişle daha önceden güncelleme yapılmış kayıtlar ve alanlarla ilgili olarak güncelleme yaparken bazı uzlaştırma işlemlerinin uygulanması gerekmektedir.

Şansımıza Delphi içinde bu konu ile ilgili olarak hazırlanmış çok kullanışlı bir diyalog bulunmaktadır. Ve ne zaman hata uzlaştırma ile ilgili bir işlem olsa, sadece istemci tarafa bu diyaloğu yerleştirmek bütün işimizi halleder. (Kendimiz de yazabiliriz, ama bu konu ile ilgili olarak diyalog kullanmak veya kendimiz yazmak, mutlaka birşeyler yapmalıyız.)

Bu diyaloğu kullanabilmek için Delphi'de File-New-Other'dan Object Repository içerisinde bulunan Reconcile Error Dialog (Hata Uzlaştırma Diyaloğu) bileşeni çift tıklama suretiyle seçilmelidir.

Bu işlemi gerçekleştirdiğimiz zaman DataSnap istemci projemize RecError.pas adında yeni bir dosya eklenecektir. Bu dosya, Hata Güncelleme diyaloğu ile ilgili tanımlamalar ve güncelleme hataları ile ilgili çözüm yollarını içeren bir dosyadır.

İhtiyaç duyulduğu zaman dinamik olarak HataUzlaştırmaFormu (ReconcileErrorForm) oluşturulacaktır. Peki bu özel formu ne zaman ve nasıl kullancağız? Aslında çok basit. Her ne sebeple olsun güncellenemeyen her kayıtla ilgili olarak TClientDataSet bileşeninin OnReconcileError olay tetikleyicisi çağrılacaktır. Bu olay tetikleyicisi aşağıda görülebilir.

```
procedure TForm2.ClientDataSet1ReconcileError(DataSet: TClientDataSet;
E: EReconcileError; UpdateKind: TUpdateKind;
var Action: TReconcileAction);
```

Bu, 4 argümanı olan bir olay tetikleyicisidir. İlki, hatayı veren TClientDataSet bileşeni, ikincisi, hata durumunun oluşması ile ilgili özel bir mesaj içeren ReconcileError (Hata Uzlaştırıcı), üçüncüsü, hatayı oluşturan güncelleme türü (UpdateKind -silme, ekleme veya güncelleme olabilir), sonuncusu da ele alınmasını istediğimiz yapılacak iştir.

Yapılacak iş olarak şu türde enum değerleri döndürülebilir. (Buradaki sıralama kendi sıralamasıdır.)

raSkip : Bu kaydı güncelleme, ama yapılmamış güncellemeleri değişim logunda tut. Bir dahaki sefere tekrar dene

raAbort : Bütün uzlaştırma olay tetiklemelerini iptal et. Daha fazla kayıt uzlaştırma için olay tetiklemesin

raMerge : Güncelleme yapılan kaydı, veritabanında güncellenmiş kayıtla birleştir, sadece değişen alanları senin tarafında da değişmişse

raCorrect : Güncellenmiş kaydı, olay tetikleyicisi içerisinde bizim yaptığımız doğru bir kayıt ile yer değiştir. (Bu Uzlaştırma diyaloğu içinde de olabilir. Kullanıcı müdahalesi olacağı zamanlarda geçerlidir bu durum)

raCancel : Bu kayıtla ilgili bütün güncellemeleri iptal et, lokalde aldığımız orjinal haline geri döndür

raRefresh : Bu kayıt ile ilgili bütün güncellemeleri iptal et, şu anki veritabanında bulunan değerleri geri yükle

Hata Uzlaştırma Formu (ReconcileErrorForm) kullanmanın avantajı, kendimizi bütün bunlarla uğraşmak zorunda bırakmamamız. Sadece 2 şey yapmamız gerekiyor. İlki, ErrorDialog unit'ini istemci formumuzda belirtmek (Use Unit özelliği ile), ikincisi ise OnReconcileError olay tetikleyicisi içerisine tek satırlık bir kod eklemek. Bu tek satır kodda ErrorDialog uniti içerisindeki HandleReconcileError metodunu çalıştırmak içindir. Bu metodun da OnReconcileError gibi 4 adet argümanı vardır. Bu yüzden argümanları birinden diğerine taşımak sıkıntılı bir işlemdir (Ne daha az, ne daha fazla). Bu yüzden kod aşağıdaki gibi olur.

```
procedure TFrmClient.ClientDataSet1ReconcileError(DataSet: TClientDataSet;
E: EReconcileError; UpdateKind: TUpdateKind;
var Action: TReconcileAction);
begin
        Action := HandleReconcileError(DataSet, UpdateKind, E)
end;
```

--- Hata Uzlaştırmalarını Göstermek ---

Önemli olan soru, bütün bunlar uygulamada nasıl çalışıyor? Bunu test etmemiz için eş zamanlı olarak 2 adet DataSnap istemci uygulaması çalıştırmamız gerekmektedir. Şu anki istemci ve sunucu uygulamalarında bu testi yapabilmek için aşağıdaki adımları takip etmemiz gerekmektedir.

DataSnap Sunucu uygulamasını başlat
İlk DataSnap istemci uygulamasını başlat, bağlan butonuna tıkla
İkinci DataSnap istemci uygulamasını başlat, bağlan butonuna tıkla.Veri şu anda zaten çalışan aynı DataSnap Sunucusundan temin edilecektir.
İlk DataSnap istemcisini kullanarak, ilk kaydın isim alanını değiştir.
İkinci DataSnap istemcisini kullanarak, ilk kaydın isim alanını değiştir.
İlk DataSnap istemci uygulamasındaki Apply Updates (güncellemeleri uygula) butonuna tıkla. Güncellemeleri sorunsuz bir şekilde yapacaktır.
İkinci DataSnap istemci uygulamasındaki Apply Updates butonuna tıkla. Bu sefer ilk istemci bu kayıt üzerinde bir değişiklik yaptığı için hata verilecek ve OnReconcileError (Hata Uzlaştırma) olay tetikleyicisi çalışacaktır.
Burada çıkan Hata Güncelleme (Update Error) diyaloğunda karşımıza çıkan seçenekleri teker teker deneyerek ne işe yaradıklarını ve aralarındaki farkları test etme imkanına sahip oluruz. Skip ve Cancel arasındaki farklara, bir de Correct, Refresh ve Merge arasındaki farklara dikkat etmek gerekiyor.

Skip bir sonraki kayda gider, o anki istenen güncellemeyi atlar. Uygulanmayan değişiklik, değişiklik logunda tutulur. Cancel da istenen güncellemeyi atlar, ama aynı güncelleme paketindeki bütün güncellemeleri iptal ederek atlar. İkisinde de ilk talep edilen güncelleme iptal ediliyor ama Skip'de diğer güncellemeler içind evam ediyor, Cancel'da ise diğer değişiklikler de iptal ediliyor.

Refresh sadece kayda yaptığımız güncellemeleri unutarak sunucudaki halini getiriyor. Merge bizde güncellenen veri ile sunucudaki veriyi birleştirmeye çalışıyor.(Bizde yapılan değişiklikleri sunucunun içerisine atarak). Refresh ve Merge içerisinde tekrar bir değişiklik talebi gitmeden sunuc ve kullanıcı arasında verinin senkronize olması sağlanıyor. (Cancel ve Skip'de değişiklik talebi tekrar gönderiliyor)

Correct, ki en güçlü çözümdür, güncellenecek kaydın ne şekilde güncelleneceğine dair size seçenek sunuyor. Ama bunun için diyalogdaki alanlara bazı değerler yazmanız gerekiyor.

--- DataSnap Veritabanı Dağıtımı ---

Veritabanı kullanan DataSnap sunucusunun dağıtımıi basit bir DataSnap sunucusunun dağıtımında biraz daha karmaşıktır. İstemci tarafta her hangi bir değişiklik yapmaya gerek yoktur. Hala thin client veya smart client bir uygulamadır ve uygulama dosyasını tek başına dağıtmak için ana formun uses kısmına MidasLab değerini eklemek yeterli olacaktır.

Sunucu için aynı zamanda veritabanı sürücülerini de dağıtmak gerekmektedir. DBX4 kullanırken TSQLConnection bileşenini kontrol ederken aynı zamanda dbxconnections.ini ve dbxdrivers.ini dosyalarının, ki bu dosyalar da XP için C:\Documents and Settings\All Users\Documents\RAD Studio\dbExpress\7.0 adresinde, Vista ve Windows 7 için C:\Users\Public\Documents\RAD Studio\dbExpress\7.0 directory on Windows Vista and Windows 7 adresinde bulunmaktadır, kontrol edilmesi gerekir.

dbxdrivers.ini dosyası verilen Driver, DriverPackageLoader ve MetaDataPackageLoader bilgilerine göre değişir. BlackFishSQl için mesela DBXClientDriver140.bpl dosyasıdır. Bu yüzden mesela BlackFish için dağıtım yapılırken detay bilgiler için RadStudio\7.0 içerisindeki deploy_en.htm dosyasına bakılması gerekir.

--- Varolan Uzak Data Modülleri Tekrar Kullanmak ---

Eğer daha önceden oluşturduğumuz TRemoteDataModule sınıflarımız varsa, bunları yeni DataSnap yapısı içerisinde tekrar kullanabiliriz. Ama sunucu tarafındaki özellikle COM tabanlı olan bazı işlevsellikleri kesmemiz gerekir.

En başta entegre edeceğimiz şey sadece bir data modül değil de çalışan bir DataSnap sunucu uygulaması ise komut istemindeki /unregister komutunu kullanarak sunucuyu kayıttan çıkarmalıyız (unregister). Eğer başlangıçtan doğru şeyi yapmazsak sonrasında uzak data modülü kayıttan çıkarma imkanımız olmaz (Eğer projenin yedeğini almadıysak)

Uzak data modülün (remote data module) kod kesiminde initialization kısmında yer alan kodları silmemiz gerekmektedir. Eğer yazılan kodun Delphi 2007 ve öncesi ya da Delphi 2009 ve sonrası için uyumlu olmasını istiyorsak aşağıdaki kod parçasını eklememiz gerekir.

```
{$IF CompilerVersion >= 20}
        initialization
                TComponentFactory.Create(ComServer,
TRemoteDataModule2010,
                Class_RemoteDataModule2010, ciMultiInstance, tmApartment);
{$IFEND}
end.
```

Ayrıca UpdateRegistry rutinini de silmemiz gerekir, eğer silmek istemiyorsak IFDEF kalıbı kullanarak da halledebiliriz.

```
{$IF CompilerVersion >= 20}
        class procedure UpdateRegistry(Register: Boolean;
```

```
                const ClassID, ProgID: string); override;
{$IFEND}
```

COM tabanlı olmayacak bir DataSnap sunucu yapısına sokmamız için yapılması gereken en önemli değişiklik, tür kütüphanelerini (type library) ve tür kütüphanesinin import edildiği unit'leri kaldırmak gerekiyor. Burayı IFDEF ile kapatmak bir çözüm değil bu yüzden Delphi 2007 ve öncesi (COM tabanlı) veya Delphi 2009 ve sonrası (COM tabanlı değil) için projenin yedeğinin alınması gerekiyor. DataSnap sunucu uygulamasında bir adet TDSServerClass bileşeni kullanmak ve daha önce de yaptığımız gibi TRemoteDataModule sınıfını döndürmemiz gerekiyor. En sonunda COM tabanlı mimaride varsayılan olarak yapılan, TRemoteDataModule içerisinde tanımlanmış bütün metodları protected diye tabir edilen bölümden alarak public alana taşımamız (yeni nesi DataSnap mimarisinde varsayılan olarak yapılan teknik) gerekiyor.

--- DataSnap Filtreleri ---

Bu bölümde filtrelerin nasıl kullanıldığını, var olan DataSnap filtrelerini nasıl kullanacağımızı (sıkıştırma gibi) ve kendi filtrelerimizi nasıl yazacağımızı anlatmaya çalışacağım. DataSnap filtreleri iletişim byte akışına (communication byte stream) dahil olan özel DLL'ler olup bir dizi filtreyi uygularlar. Bu yüzden sıkıştırma ve kriptolamayı veya loglama ve sıkıştırmayı birlikte kullanabiliriz.

DataSnap sunucu ve istemcilerinde filtreleri özelleştirmek için 2 yer bulunmaktadır. Sunucu tarafında TDSTCPServerTransport bileşeni içerisindeki Filtreler (Filters) özelliğinde biz dizi filtre tanımlanabilir. Ve istemci taratfa da DataSnap istemci proje dosyasının uses bölümünde bir dizi filtre değeri eklenebilir. Her DataSnap filtresi için otomatik kayıt olduğundan, bu istemci taraf için yeterli olan bir eylemdir.

OnConnect olay tetikleyicisinde bağlantı için kayıtlı filtreleri şu şekilde kullanabiliriz. (Mesela LogInfo gibi bir fonksiyonu bağlantı bilgisini log dosyasına kaydetme için)

```
procedure TServerContainer1.DSServer1Connect(
DSConnectEventObject: TDSConnectEventObject);
var
        i: Integer;
begin
        LogInfo('Connect ' + DSConnectEventObject.ChannelInfo.Info);
        for i:=0 to DSConnectEventObject.Transport.Filters.Count-1 do
                LogInfo('   Filter: ' +
                DSConnectEventObject.Transport.Filters.GetFilter(i).Id);
end;
```

--- ZLibCompression Filtresi (Sıkıştırma) ---

Örnek olarak Delphi 2010 içerisinde hazır olarak var olan, istemci ve sunucu arasındaki veri akışını sıkıştırmak için kullanılan bir filtre verilebilir. Bunun için örnek olarak DbxCompressionFilter unit dosyası içerisinde bulunan ZLibCompression filtresi verilebilir.

TCP/IP için olan TDSTCPServerTransport ve HTTP için olan DSHTTPService bileşenlerinin ikisinin de TTransportFilterConnection nesnesi bulunduran bir Filtre (Filters) özelliği bulunmaktadır. Filters özelliğine çift tıklama suretiyle filtre

kolleksiyonunu düzenleyebiliriz. Burada teni bir TTransportFilterItem nesnesi ekleyip Object Inspector kullanarak ID ve diğer bilgilerini düzenleyebiliriz. İlk izlenim olarak FilterID özelliğinden düzenlenecek ve atanacak şekilde Delphi 2010 ZLibCompression filtresine sahiptir.

Unutmamak gerekir ki sunucu tarafta Filter özelliğine gerekli atamaları yaptıktan sonra istemci tarafta bu filtreyi kullanmak istediğimizi belirtmeliyiz. (Sunucudan gidecek cevapları sıkıştırma ve istemciden gelen sıkıştırılmış talepleri açma gibi) Bunun içi yapmamız gereken tek şey istemci formunda (ClientForm) uses kısmına DbxCompressionFilter unit dosyasını eklemektir. Bu TTrapsportFilterServer nesnesini otomatik olarak istemciye kaydedecek ve sunucu ile iletişim kurduğundan emin olacaktır.

Eğer uses kısmına gerekli eklemeyi yapmazsak istemci uygulamayı çalıştırdığımızda "İletişim filtresi olan ZLibCompression kayıt olmamıştır. Sunucu ile iletişim için önce filtre sınıfının kayıt edilmesi gerekmektedir" içerikli bir hata mesajı alınacaktır.

--- Log Filtresi (Loglama) ---

Delphi 2010 DataSnap yapısı kendi filtrelerimizi oluşturmamıza imkna tanımaktadır. Bunu TTrasnportType sınıfından bir sınıf türeterek yapabiliriz. Bu yeni sınıfta ana sınıfın metodlarını yeniden yazarak (override) tanımlamalar yapabiliriz. Örnek olarak aşağıdaki gibi bir TLogFilter adında loglama için kullanılacak bir sınıf oluşturabiliriz.

```
unit LogFilter;
interface
uses
	SysUtils, DBXPlatform, DBXTransport;

type
TLogFilter = class(TTransportFilter)
	private
	protected
		function GetParameters: TDBXStringArray; override;
		function GetUserParameters: TDBXStringArray; override;
	public
		function GetParameterValue(const ParamName: UnicodeString):
UnicodeString; override;
		function SetParameterValue(const ParamName: UnicodeString;
		const ParamValue: UnicodeString): Boolean; override;
		constructor Create; override;
		destructor Destroy; override;
		function ProcessInput(const Data: TBytes): TBytes; override;
		function ProcessOutput(const Data: TBytes): TBytes; override;
		function Id: UnicodeString; override;
end;

const
LogFilterName = 'Log';
```

Bu sınıfın kod kesimi içerisindeki pek çok alan boş olacaktır, sadece ProcessInput ve ProcessOutput metodları uygulanırken gönderilecek log verisinin filtrelenmesi için

gerekli metodlar dolrurulacaktır, gerisi boş bırakılabilir. Boş olmayacak metodlar aşağıda belirtilmiştir.

```
	function TLogFilter.SetParameterValue(const ParamName, ParamValue:
UnicodeString): Boolean;
	begin
		Result := True;
	end;

	constructor TLogFilter.Create;
	begin
		inherited Create;
	end;

	destructor TLogFilter.Destroy;
	begin
		inherited Destroy;
	end;

	function TLogFilter.ProcessInput(const Data: TBytes): TBytes;
	begin
		Result := Data; // log incoming data
	end;

	function TLogFilter.ProcessOutput(const Data: TBytes): TBytes;
	begin
		Result := Data; // log outgoing data
	end;

	function TLogFilter.Id: UnicodeString;
	begin
		Result := LogFilterName;
	end;
```

En sonunda DataSnap iletişim filtresinin kod kesimindeki en önemli noktalardan biri, initialization ve finalization kısımlarındaki kayıt olma bölümüdür. Buradaki kod aşağıda belirtilmiştir.

```
	initialization
		TTransportFilterFactory.RegisterFilter(LogFilterName, TLogFilter);
	finalization
		TTransportFilterFactory.UnregisterFilter(LogFilterName);
	end.
```

Bu iletişim filtresini kullanabilmek için TDSTCPServerTranport veya DSHTTPService bileşenlerindeki Filters özelliğine bu filtreyi eklememiz gerekir. Tasarım zamanında şu anda ZLibCompression filtresi dışında her hangi bir filtre bulunmamaktadır örneğimizde. (Biz eklemediğimiz sürece) Şansımıza, iletişim filtrelerini çalışma zamanında da eklememiz mümkündür. Bunu ServerContainerUnitDemo unit dosyasının uses kısmına filtre unit'ini ekleyerek ve manuel olarak filtrenin adını filtre listesine ekleyerek yapabiliriz. Bunla ilgili örnek kod aşağıdadır.

```
procedure TServerContainer1.DataModuleCreate(Sender: TObject);
begin
        DSTCPServerTransport1.Filters.AddFilter(LogFilterName);
        DSHTTPService1.Filters.AddFilter(LogFilterName);
        DSHTTPService1.Active := True;
end;
```

Sunucunun Log filtresini kullandığından emin olmamız ve aynı şekilde istemci tarafta da ilgili unit dosyasının uses kısmına bu filtre unit dosyasının eklendiğinden emin olmamız gerekmektedir. Eğer bunları yapmazsak aynı şekilde bir hata mesajı alırız.

Unutmamak gerekir ki her uygulama (DataSnap Sunucu ve İstemci uygulamaları) kendi log dosyasını tutar, bu yüzden aynı loglama filtresi kullanılmasına rağmen, ParamStr(0) daki gibi hangi hedefin log mesajını ürettiğine dair bir bilgi eklemek zorunda değiliz.

--- Encryption Filtresi (Kriptolama) ---

Bir üst örnekte (log filtresi) de belirtildiği gibi bu örneği genişletip kendi filtrelerimizi daha kompleks bir şekilde genişletebiliriz. DataSnap filtreleri (hazır olarak gelenler) karmaşık değildir. Şu da bir gerçektir ki 3. parti filtreler de kullanılabilir ve Daniele Teti tarafından yapılan, içerisinde 3 gruba ayrılmış bir şekilde Delphi 2010 DataSnap için yapılmış 9 tane ek filtre bulunduran DataSnap Filtre Özetleri (DataSnap Filters Compendium) http://www.danieleteti.it/?p=168 adresinde bulunabilir. Hash grubu; MD5, MD4, SHA1 ve SHA512'yi, Cipher grubu; BlowFish, RijnDael, 3TDES ve 3DES'i, Compress (Sıkıştırma) grubu da LZO'yu destekler. Bütün kaynak kodlarına da erişilebilir.

--- DataSnap Web Hedefleri ---

Windows hedefleri dışında ISAPI, CPI veya Web App Debugger türünden hedefler oluşturmaya yarayan bir sihirbaz da bulunmaktadır. Öncelikle bu hedeflerin her birinin yararlarından bahsedeceğim, sonrasında tek bir proje grubu içerisinde Windows hedeflerinde yaptığım gibi bu farklı web hedeflerini tek tek oluşturup aynı dosyaları paylaşmalarını sağlayıp, sonuç olarak tek bir proje grubu içerisinde 3 ayrı proje oluşturarak aynı DataSnap sunucu nesnesini kullanan 3 ayrı uygulama geliştirmiş olacağım.

DataSnap sunucu uygulamaları çok iyi bir şekilde çalışmasına rağmen, bu sunucu uygulamaları dağıtma kabiliyeti olmadığı zamanlar da oluyor. Mesela istemcilerin sunucuya bağlanması için firewall üzerindeki gerekli portları açamama durumu olabiliyor. Şansımıza bu tarz olayların olduğu durumlarda, bir web sunucusu (web server) üzerinde bir web sitesi tanımlanabilir, bu yüzden 80 portu genelde açıktır (web servis için). Microsoft Internet Information Server (IIS - İnternet Bilgi Sunucusu)'ı web server olarak kullanabileceğimiz için Delphi 2010'daki DataSnap bölümünde DataSnap WebBroker Application bölümünü IIS üzerinden dağıtılacak bir proje geliştirmek üzere kullanabiliriz.

DataSnap WebBroker Application sihirbazı, bize 3 seçenek sunar. Bunlardan biri, gerçek bir WebBroker uygulaması olmayıp, web uygulaması istemci debugger'ı (Web App Debugger) olarak sadece debug amaçlı olarak kullanılır. Web App Debugger istemcisi oldukça güçlü bir araçtır, bu araç bize Delphi IDE'si üzerindeki Tool (Araçlar) menüsündeki Web App Debugger'i, DataSnap Web App Debugger Client uygulamasını debug ederken bir host olarak kullanmamıza izin verir. Bir CGI veya ISAPI/NSAPI web

uygulamasını debug etmek (hata ayıklamak) daha zor olduğundan eğer uygulama hala geliştirme aşamasındaysa Web App Debugger gayet güçlü bir çözümdür.

ISAPI/NSAPI DLL'i ve CGI uygulama hedefleri gerçek DataSnap Sunucu projeleri için kullanılır.

Unutmamak gerekir ki, CGI uygulaması geliştirmek o kadar da iyi bir fikir değildir, çünkü bu uygulama her gelen talep için yüklenir ve boşaltılır. Bu da veritabanına bağlanıp iş yapmak için ek zaman demektir ki, bu uygulamanın kullanıldığı uygulamanın performansı ile ilgili olarak size bir fikir verebilir. DLL dosyası ile sonuçlanacak bir ISAPI hedefinde, DLL sadece bir kere yüklenir, ve herhangi bir performans düşüşüne neden olmayacak şekilde ardarda gelen istekler (farklı kullanıcılar için de olabilir) boyunca bellekte yüklü kalmaya devam eder. ISAPI DLL'inin en büyük dezavantajı, yer değiştirmesinin zor olmasıdır (eğer web server'a tek erişim yolu FTP ise) ama bu sorunu çözmek için pek çok ISAPI Manager bulunmaktadır (Detaylar için web host sağlayıcısı ile görüşün).

ISAPI DLL hedefinin bir diğer dez avantajı, debug edilmesinin (hata ayıklama) zor olmasıdır. Bunun için öncelikle IIS'i host uygulama olarak yüklemek gerekir ki bu her zaman planlandığı gibi çalışmaz. Ama Web App Debugger uygulaması bu sorunu çözer. Sadece yapılması gereken iş aynı proje grubu içerisinde 2 proje oluşturmak, ki bunlar aynı DataSnap metod ve kodlarını kullanacak. Bu ilk demo için güzel bir başlangıç olur, ve üzerine gerçek hayatta kullanılacak bazı teknikler de eklenerek ortaya çalışan bir iskelet çıkarabiliriz.

--- Web App Debugger Hedefi (Web Uygulaması Hata Ayıklayıcı) ---

Öncelikle karşımızdaki sihirbazdan Web App Debugger executable seçeneğini seçeriz, sonra mesela DSWAD gibi bir sınıf ismi belirleriz ve Support HTTP Authentication (HTTP Yetkilendirmesini Destekle) seçeneğini işaretleriz.

Tamam (OK) dediğimiz zaman, 3 unit dosyasına sahip yeni bir proje oluşturulur. Eğer proje dizinimizde Project1 veya Unit1 gibi isinlendirilmiş dosyalar yoksa projemizin adı Project1 unit dosyalarımızın adı da Unit1, ServerMethodsUnit1 ve Unit2 olarak adlandırılmış bir şekilde oluşturulur. İlk unit dosyası boş bir formdur, bu Web App Debugger uygulamasına mahsus bir dosyadır, ve diğer web hedefleri tarafından kullanılmasına gerek yoktur. Bu unit dosyasını WADForm.pas olarak kaydedelim. İkinci dosyamız ServerMethodsUnit1.pas dosyasıdır ve diyalogda belirtildiği üzere TDSServerModule tarafından türetilmiş sunucu modülümüzü içerir. Bu unit dosyasına sonra döneceğiz ama şimdilik ServerMethodsUnit1.pas olarak kaydedelim. Sonuncu unit dosyamız üzerinde 4 adet bileşen bulunduran (Support HTTP Authentication özelliğini seçmeseydik 3 olacaktı) web modülü dosyamızdır. Bu dosya da gelen talepleri almak ve projedeki DataSnap sunucu modülünde göstermek için kullanılır. Bu dosyayı da DSWebMod.pas olarak kaydedelim.

En sonunda proje dosyasını da DSWADServer.dproj olarak kaydedelim ki bu projenin bir DataSnap Web App Debugger projesi olduğu anlaşılsın.

--- ISAPI Hedefi ---

ServerMethodsUnit1.pas ve DSWebMod.pas dosyalarını modifiye etmeden ve devam etmeden önce yeni bir projeyi proje grubuna eklemeliyiz. Bu ekleyeceğimiz proje ISAPI/NSAPI Dynamic Link uygulama projesi olacaktır. Proje grubuna sağ tıklayıp Yeni Proje ekle dedikten sonra DataSnap WebBroker Application sihirbazından ISAPI/NSAPI Dynamic Link Application seçeneğini seçmeliyiz. Bu sefer daha önceden oluşturduğumuz DSWADServer projesnin unit dosyalarını kullanmak istiyorsaki alt

taraftaki seçeneklere dokunmamamız bizim için daha iyi olur. Böylece projeler gerekli olan aynı unit dosyalarını kullanabilir.

Tamam dediğimiz zaman yeni bir projeyi (Project1.dproj) , ServerMethodsUnit2.pas ve Unit.pas dosyaları ile birlikte oluşturur. Biz burada oluşturulan ServerMethodsUnit2.pas ve Unit.pas dosyaları yerine daha önceden oluşturduğumuz ServerMethodsUnit1.pas ve DSWebMod.pas'ı kullanmak istiyoruz. Bunun için ServerMethodsUnit2.pas dosyasına sağ tıklayıp silmeliyiz (Çıkan diyaloğa evet dedikten sonra dosyayı kaydetmediysek zaten sorun yok). Aynı işlemi Unit1.pas için yaptıktan sonra ISAPI projesine sağ tıklayarak daha önceden oluşturulan ServerMethodsUnit1.pas ve DSWebMod.pas dosyalarını projeye ekleriz. En sonunda Project1.dproj dosyasının adını DSISAPIServer.dproj olarak değiştirerek kaydedip işlemleri bitirmeliyiz.

Şu anda bir proje grubu içerisinde aynı dosyaları ServerMethodsUnit1.pas ve DSWebMod.pas dosyalarını paylaşan 2 adet projemiz bulunmaktadır.

Bu ayarlama 2 farklı projeyi derlememize olanak tanımaktadır. DSWADServer projesini test ve debug (hata ayıklama) hedefi olarak, DSISAPIServer projesini de DataSnap sunucusunun IIS üzerinden dağıtılması için derleriz.

ServerMethodsUnit1 dosyasına ilgili web metodlarını eklemeden önce ISAPI/NSAPI projesindeki TDSServerModule tarafından proje dosyasının içerisinde otomatik olarak oluşturulan kod kesimini silerek düzenleme yapmalıyız. TDSServerModule de aslında bir TDataModule olduğundan, yani bir web modülüdür de aynı şekilde, ISAPI DLL'ini çalıştırmaya kalktığımızda bir web broker uygulamasında sadece bir web modülü olacağından hata mesajı verecektir. ISAPI projesinin kaynak kodunu açarak ana begin-end bloğunda aşağıdaki gibi bir değişiklik yapmalıyız.

```
begin
        CoInitFlags := COINIT_MULTITHREADED;
        Application.Initialize;
        Application.CreateForm(TWebModule2, WebModule2);
        //   Application.CreateForm(TServerMethods1, ServerMethods1);
        Application.Run;
end.
```

Bu düzenleme, tek data modülüne izin verilmesinden dolayı oluşacak hata mesajını engeller. Unutmamak gerekir ki bu hata mesajını yayınlanmış, dağıtılmış ISAPI DLL'ini çağırdığımızda göremeyebiliriz, ama sunucunun zaman aşımı ya da hata durumlarında bu durumu hatırlamak önemlidir.

--- Sunucu Metodları, Dağıtım ve ıstemciler ---

Fonksiyonellik eklerken, bütün hedefler tarafından paylaşılan ServerMethodsUnit1.pas dosyası üzerinde çalışmalıyız. Varsayılan olarak tıpkı Windows DataSnap sunucularındaki gibi bir metod otomatik olarak oluşturulmuştur, ama biz daha önceki bölümlerde işlediğimiz 2 metodu da ekleyeceğiz. Sunucu metodları yazıldığı zaman biz ISAPI DLL'ini Microsoft'un IIS', gibi bir web server gibi dağıtıp kullanıma açabiliriz. Bu JimTerney'in http://blogs.embarcadero.com/jimtierney/2009/08/20/31502 adresindeki makalesinde detaylı olarak anlatıldığından tekrar etmeyeceğim

Şu anda dağıtım için bir web sunucunuz olmadığından benim web sunucumdan dağıtılmış DataSnap ISAPI DLL'i ile istediğiniz gibi oynayabilirsiniz. Unutmamak gerekir ki ben bir TDataSetProvider göstermedim, ve GetEmployees metodu ile herhangi bir data döndürmüyorum, ama ServerTime ve EhcoString metodları gayet

güzel bir şekilde çalışıyor ve bu sunucuya karşılık olarak bir istemci yazabilmeniz için yeterlidir bu özellikler.

ISAPI DataSnap sunucusuna bir istemci programının içinden bağlanmadan önce bu sunucuya bağlantı kurup kurmayacağımız görmek için Data Explorer kullanmamız gayet iyi bir fikirdir. Data Explorer (IDE içerisinde bulunan özellik), DATASNAP adında yeni bir kategori içermektedir ve burayı açtığımızda DATASNAPCONNECTION adı verilen ilk uygun bağlantı görünmektedir. Bunun üzerinde değişklik yapmak için sağ tıklayarak Modify (Düzenle, değiştir) seçeneğini seçmemiz gerekir.

Bu diyalogda Protokol, Host (Eğer kendi web sunucunuz (web server) yoksa www.bobswart.nl adresini kullanabilirsiniz.), Port, sunucudaki ISAPI DataSnap Sunucu uygulamasının URL olarak yolu ki bu cgi-bin/DSISAPIServer.dll oluyor gibi özelliklerini değiştirip gerekli düzenlemeleri yapabiliriz. Her şeyin doğru çalıştığından emin olmak için Test Connection butonuna tıklamalıyız.

Artık tamam (OK) diyerek diyaloğu kapatabilir ve DATASNAPCONNECTION hücresini genişleterek Tablo (Table), Görüntü (View), Prosedür (Procedure), Fonksiyon (Function) ve Eş Anlamlıları (Synonym) görebiliriz. Görüldüğü gibi Prosedür bölümünde DSAdmin, DSMetaData, TServerMethods1.AS_xxx metodlarını ve bizim oluşturduğumuz EchoString, ServerTime ve GetEmployee prosedürlerini görmek mümkündür.

Herhangi bir DataSnap istemci uygulaması geliştirmeye gerek kalmadan buradaki metodların bazılarını test edebiliriz. Mesela EchoString (bizim gönderdiğimiz neyin geri döndüğünden emin olmak için) prosedürünü test edebiliriz. GetEmployees prosedürüne sağ tıklayarak View Parameters (Parametreleri Göster) seçeneğine tıkladığımız zaman IDE içerisinde bizim Value (Değer) parametresini girebileceğimiz bir pencere açılır (Mesela 42 gireriz). Sonrasında pencereye sağ tıklayarak Execute (çalıştır) dediğimiz zaman ortaya çıkacak sonuç ReturnValue (Geri dönüş değeri) alanında gösterilir.

Bu bize DataSnap sunucusundaki sunucu metodlarını çağırabildiğimizi ispatlar. DataSnap istemci uygulamasından sunucuya bağlanmak istersek, sadece TSQLConnection bileşeninin özelliklerini modifiye etmemiz gerekir. Daha önceden Windows DataSnap sunucusuna bağlanmıştık, ama bu ayarları DataSnap Web sunucusu için güncellemek gerekiyor.

Unutmamanız gerekir ki benim web sunucumdaki DSISAPIServer.dll dosyasını kullanmak isterseniz, TDataSetProvider bileşeni erişilmez olarak düzenlediğim ve GetEmployees metodunda herhangi bir veri döndürmediğim için kullanamazsınız. Ama sorunsuz bir şekilde EchoString ve ServerTime metodlarını kullanabilirsiniz.

--- REST ve JSON ---

DataSnap 2010 REST ve JSON teknolojilerine destek vermektedir. DataSnap 2010'un DataSnap HTTP istekleri için REST desteği bulunmaktadır. Mesela DataSnap sunucusunun URL adresi http://www.bobswart.nl/cgi-bin/DSISAPIServer.dll ise, sonra biz bu URL adresine /datasnap/rest ekleyerek sonuna da Sunucu Metod sınıfının da adını, metodu ve argümanlarını da ekleyebiliriz. İdeal hali şu şekildedir.

http://server/datasnap/rest/<class>/<method>/<parameters>

Benim sunucumdaki TServerModule1 sınıfı içerisindeki ServerTime metodu için gerekli olan URL adres örneği şu şekildedir.

http://www.bobswart.nl/cgi-bin/DSISAPIServer.dll/datasnap/rest/TServerMethods1/ServerTime

REST uyumlu bu URL'yi çağırarak, sonucu JSON sonucu olarak döndürdüğümüzde şöyle bir sonuç elde ederiz.

{"result":["2009-10-16 16:01:33.145"]}

Marco Cantu, Delphi 2010 ve REST istemcileri ile ilgili belgesinde bu konu üzerine detaylı olarak eğiliyor.

--- Geri Çağırma ---

DataSnap sunucularına yapılan REST uyumlu çağrıların dışında, JSON geri çağırma metodlarının yazımında da kullanılmaktadır. DataSnap 2010 mimarisi, sunucu metodun içeriğinde çalıştırılacak istemci taraflı geri çağırma desteği sunmaktadır. Bu demektir ki sunucu bir metodun çalıştırılması sırasında (istemci taraftan çağrılan), sunucu, istemci tarafından sunucu metoda bir argüman olarak gönderilecek bir geri çağırma metodu çağırabilir.

Örnek olarak hadi EchoString metodunu içine bir geri çağırma fonksiyonu eklemek suretiyle değiştirelim. EchoString'in yeni hali şu şekilde olacaktır.

```
function EchoString(Value: string; callback: TDBXcallback): string;
```

TDBXcallback türü, DBXJSON unit dosyasında tanımlıdır. Yeni EchoString metodunun kodunu yazmadan önce öncelikle geri çağırma metodunun istemci tarafta nasıl tanımlandığını görmek gerekir.

İstemci tarafta, TDBXcallback sınıfından türeyen yeni bir sınıf oluşturmak ve Execute metodunu tekrar yazmak (override) gerekir.

```
type
TCallbackClient = class(TDBXCallback)
        public
                function Execute(const Arg: TJSONValue): TJSONValue;
override;
end;
```

Execute metodunun içinde Arg adındaki argümanı esas içeriğe elimizi uzatacak ve klonyalacak şekilde TJSONValue türünden aldık. Execute metodu zaten TJSONValue türünden bir sonuç döndürüyor, ben de sadece aynı değeri döndüreceğim.

```
function TCallbackClient.Execute(const Arg: TJSONValue): TJSONValue;
var
        Data: TJSONValue;
begin
        Data := TJSONValue(Arg.Clone);
        ShowMessage('Callback: ' +
TJSONObject(Data).Get(0).JSonValue.value);
        Result := Data
end;
```

Bu örnek için geri çağırma metodu, bize EchoString metoduna gönderilen argüman değerini (metoddan da dönen değer bu), metoddan değer geri dönmeden önce, metodun çalışması sırasında döndürecektir. Sunucu taraftaki EchoString metodunun kodunda TJSONObject nesnesi içine bir string değerini argüman olarak göndererek ve bu değeri geri çağırma metoduna göndererek gerekli düzenlemeyi yapmalıyız. Kod şu şekilde olacaktır.

```
function TServerMethods2.EchoString(Value: string; callback: TDBXcallback):
string;
	var
		msg: TJSONObject;
		pair: TJSONPair;
	begin
		Result := Value;

		msg := TJSONObject.Create;
		pair := TJSONPair.Create('ECHO', Value);
		pair.Owned := True;
		msg.AddPair(pair);
		callback.Execute(msg);
	end;
```

Unutmamak gerekir ki geri çağırma metodu istemci tarafından çağrılır, ve sunucu tarafta işletilen metodun işletimi bitmeden bize istediğimiz değeri döndürür.

En sonunda, istemci tarafta çağrılan EchoString metodunun çağrılma şekli de değiştirilerek, TCallbackClient nesnesinden bir varlığı 2. argüman olarak metoda göndermek, işlem bitirilir.

```
	var
		MyCallback: TCallbackClient;
	begin
		MyCallback := TCallbackClient.Create;
		try
			Server.EchoString(Edit1.text, MyCallback);
		finally
			MyCallback.Free;
		end;
	end;
```

Bu örnek DataSnap 2010 içerisinde istemci taraflı geri çağırma metodlarının nasıl kullanılacağını gayet güzel bir şekilde göstermektedir.

--- DataSnap ve .NET ---

Delphi Prism 2010 daha önce oluşturduğumuz Win32 sunucuları için, DataSnap .NET istemcisi oluşturmak için kullanılan bir araçtır. Delphi Prism 2010 DataSnap istemcisi oluşturmak için DataSnap sunucusunun çalıştığından emin olmamız gerekir ki tasarım zamanlı olarak sunucuya bağlanabilelim.

Delphi Prism 2010'u başlatıp, View Menüsünden Server Explorer seçeneğini seçerek, Delphi Prism'in Server Explorer'ını görüntüleyelim. Burada DataSnap sunucusu ile çalışıp çalışamayacağımızı doğrulamak adına bir bağlantı oluşturmamız gerekiyor.

Server Explorer veri bağlantıları ile ilgili bir ağaç yapısına sahiptir. Data Connections (Veri Bağlantıları) hücresine sağ tıklayıp Add Connection (Bağlantı Ekle) seçeneğini seçtiğimiz zaman çıkan listeden DataSnap seçeneğini seçmemiz gerekiyor. (Eğer daha önceden seçilmiş bir tane varsa Change (Değiştir) demeliyiz.).

Sadece DataSnap veri bağlantıları ile çalışmak istemiyorsak "Always use this selection" (Her zaman bu seçimi kullan) checkbox'ındaki seçimi kaldırırız.

Devam butonuna tıklayarak diyaloğun bir sonraki sayfasına geliriz. Burada DataSnap sunucusuna bağlanmak ile ilgili olarak detay bilgiler bulunmaktadır. Protokol combobox'ından TCP/IP veya HTTP seçeriz. Sonra Host bilgisini gireriz (DataSnap sunucusunun çalıştığı makinanın adı, bu eğer aynı makina üzerinde çalışıyorsanız localhost olur). Ve Port numarası bilgisini gireriz. Bu HTTP için 80 ve TCP/IP için 211'dir ama eğer bu makaleyi okuduysanız bu iki değeri de farklı olarak girmeniz gerekiyor, en azından ServerContainerUnitDemo unit dosyasındaki iletişim bileşenlerinde verilen değerlerin aynısı olması gerekiyor. (Güvenlik sebeplerinden dolayı, daha önce bahsedildi)

Sonraki özellik Path'dir (Yol). Eğer Web Broker tabanlı bir DataSnap sunucusuna bağlanacaksak burası oldukça önemlidir. (DataSnap web server'ının URL olarak yolunu girmemiz gerekir, http://..../ domain bölümünden sonraki bölümdür).

En sonunda eğer DataSnap sunucusu HTTP yetkilendirme kullanıyorsa, kullanıcı adı ve şifre bilgilerinin de girilmesi gerekiyor.

Test Connection butonuna tıklayarak belirtilen DataSnap Sunucusuna bağlantı yapıp yapamayacağımızı test etmiş oluruz. Eğer "Test connection succeeded" (Test bağlantısı başarılı) uyarısı verirse her şeyi doğru yapmışız demektir.

Tamam dediğimiz zaman Veri Bağlantıları ağacına, DataSnap bölümüne yeni bir eleman eklenir. Bu olayda eklene hücre localhost hücresidir. Bu hücreyi genişlettiğimiz zaman Table (Tablo), View (Görüntü), ve Stored Procedure (Depolanmış Prosedür) leri görebiliriz. Table ve View bölümleri boştur, ama Stored Procedure bölümünde DataSnap sunucusu içinde gösterimine izin verilen bütün Sunucu Metodları, ki buna EchoString, ServerTime ve GetEmployee de dahildir, görülür.

Server Explorer tarafından bazı sunucu metodları test etme imkanımız bulunmaktadır. Mesela EchoString metoduna sağ tıklayıp View Parameters (Parametreleri göster) özelliğini seçeriz. Burada Value (değer) parametresini girebileceğimiz yeni bir diyalog penceresi açılır. Parametreye 42 değerini atadıktan sonra pencereye sağ tıklayıp Execute (çalıştır) deriz. Bu şekilde EchoString metodunu sunucu tarafından girilen 43 değeri ile çalıştırarak sonucu stored procedure penceresinden gösterir.

Bu, güzel olmakla birlikte GetEmployees metodunu kullanarak Employees tablosundaki verileri çağırıp kullanmak bizim için daha öğretici olur. Bu Stored Procedure'ün parametresi olmamasına rağmen biz gene de View Parameters kısmını çalıştırabiliriz ve bu bize boş bir parametre listesi verir. Bu pencereye sağ tıklayıp çalıştır dediğimizde ise bu sefer GetEmployees metodundan da döneceği gibi karşımıza çıkacak sonuç bütün Employees tablosunun kayıtları olur.

--- WinForms ıstemci (VCL Form) ---

DataSnap sunucu metodlarıyla Server Explorer üzerinden çalışmak eğlenceli olsa da bunu bir .NET uygulamasından çağırmak daha kullanışlı olur. Son örnek için File menüsünden New Project seçeneğini seçerek karşımıza çıkan projelerden Windows Application seçeneğini seçeriz. Çıkan proje WindowsApplication1 olmakla beraber bunu DataSnapClient olarak yeniden isimlendiririz.

Tamam dedikten sonra yeni bir DataSnapClient projesi ana formu için bir Main.pas dosyası ile birlikte Delphi Prism IDE'si tarafından oluşturulur.

Server Explorer'dan daha önceden oluşturduğumuz bağlantıyı seçeriz. Properties (Özellikler) kısmında bu nesneni ConnectionString (bağlantı string) de dahil olmak üzere özelliklerini görebiliriz ki ConnectionString de şu şekilde olur.

```
communicationprotocol=http;hostname=localhost;port=8080;dsauthenticatio
nuser=Bob;ds
authenticationpassword=Swart
```

Oluşturduğumuz bağlantının ait olduğu hücreye sağ tıklayarak "Generate Client Proxy" (İstemci Proxy'si oluştur) seçeneğini seçeriz ve TServerMethods1Client sınıfının kodunun bulunduğu, ki içinde EchoString, ServerTime ve GetEmployee de bulunur, bir ClientProxy1.pas dosyası otomatik olarak oluşturulur. Bu kodun görünümü aşağıdaki gibidir.

```
TServerMethods1Client = class
public
        constructor (ADBXConnection: TAdoDbxConnection);
        constructor (ADBXConnection: TAdoDbxConnection; AInstanceOwner:
Boolean);
        function EchoString(Value: string): string;
        function ServerTime: DateTime;
        function GetEmployees: System.Data.IDataReader;
```

Proxy sınıfının dışında projenin References (referans) bölümüne de bir çok referans otomatik olarak eklenir. Borland.Data.AdoDbxClient ve Borland.Data.DbxClientDriver gibi.

Görebileceğiniz gibi TServerMethods1Client sınıfının kod kesimi içerisinde 2 adet yapıcı bulunmaktadır. İkisi de ADBXConnection parametresi alır, ikincisinde bir de AInstaceOwner adında Boolean türünde bir parametre bulunur. Bu demektir ki yapıcı metodu bir argümanla çağırmamız gerekmektedir. Ve bunu desteklemek için proje ayarlarında bazı değişiklikler yapmamız gerekmektedir. Solution Explorer (Çözüm Penceresi - .NET'te proje grubuna çözüm deniyor) da DataSnapClient hücresine sağ tıklayarak Özellikler seçeneğini seçeriz. Buradaki Compatibility (Uygunluk) sekmesindeki "Allow create constructor calls" (Yapıcı metod çağrılarını oluşturmaya izin ver) seçeneğini seçeriz. Bu sayede otomatik oalrak Create yapıcısı yeni bir operatör kullanmak yerine argümanları ile birlikte otomatik olarak oluşturulur.

Artık ana forma dönüp bir buton yerleştirebiliriz. Butonun Click olay tetikleyicisi içerisinde DataSnap sunucusuna bir bağlantı kurarak, sunucu metodlarından birini çağırabiliriz. İlgili kod aşağıdadır.

```
method MainForm.button1_Click(sender: System.Object; e:
System.EventArgs);
var
        Client: ClientProxy1.TServerMethods1Client;
        Connection: Borland.Data.TAdoDbxDatasnapConnection;
begin
        Connection := new Borland.Data.TAdoDbxDatasnapConnection();
        Connection.ConnectionString :=

'communicationprotocol=http;hostname=localhost;port=8080;dsauthenticatio
nuser=Bob;dsauth
```

```
            enticationpassword=Swart';
            Connection.Open;
            try
                Client :=
ClientProxy1.TServerMethods1Client.Create(Connection);
                MessageBox.Show(
                Client.EchoString('Delphi Prism 2010'));
            finally
                Connection.Close;
            end;
        end;
```

Sonuç da "Delphi Prism 2010" stringidir ve görülebilir.

Benzer bir yöntemle GetEmployees metodu da çağrılıp sonuçlar bir DataGridView (.NET gridi) içerisine atılabilir. Yalnız burada bir sıkıntı vardır ki, bu metoddan dönen değer bir IDataReader (TSQLDataSet'in sonucuna eşdeğer bir nesne) olup, bir DataSet veya DataTable değildir. Bu yüzden GetEmployees metodunun sonucunu bir DataSet (Win32 tarafında TClientDataSet'e eşdeğer bir nesne) içerisindeki bir DataTable'a aktarmak için biraz daha kod eklememiz gerekmektedir.

```
        method MainForm.button1_Click(sender: System.Object; e:
System.EventArgs);
        var
            Client: ClientProxy1.TServerMethods1Client;
            Connection: Borland.Data.TAdoDbxDatasnapConnection;
            Employees: System.Data.IDataReader;
            ds: System.Data.DataSet;
            dt: System.Data.DataTable;
        begin
            Connection := new Borland.Data.TAdoDbxDatasnapConnection();
            Connection.ConnectionString :=

        'communicationprotocol=http;hostname=localhost;port=8080;dsauthenticatio
nuser=Bob;dsauth
            enticationpassword=Swart';
            Connection.Open;
            try
                Client :=
ClientProxy1.TServerMethods1Client.Create(Connection);
                Employees := Client.GetEmployees;
                ds := new DataSet();
                dt := new DataTable("DataSnap");
                ds.Tables.Add(dt);
                ds.Load(Employees, LoadOPtion.PreserveChanges,
ds.Tables[0]);
                dataGridView1.DataSource := ds.Tables[0];
                MessageBox.Show(
                Client.EchoString('Delphi Prism 2010'));
            finally
                Connection.Close;
            end;
        end;
```

Sonuç da Delphi Prism WinForms uygulaması üzerinde bir DataGridView üzerinde gösterilir ki, bu da bize DataSnap sunucularına bağlanmak için .NET istemcileri geliştirebileceğimiz gösterir.