



EMBARCADERO AKADEMIJA

Embarcadero Akademija 2013: Embarcadero RAD Studio in baze podatkov

Primož Gabrijelčič
<http://thedelphigeek.com>

Kazalo

Uvod	2
DataSnap	3
Zgodovina	3
DataSnap v RAD Studiu XE4.....	3
Večnivojski dostop do baze podatkov.....	4
Strežnik	4
Odjemalec.....	6
Obravnavanje napak.....	8
Avtentikacija.....	9
Filtri.....	9
Življenski cikel strežniškega razreda.....	9
Poizvedbe s parametri.....	10
Strežnik	10
Odjemalec.....	10
DataSnap in REST.....	11
Strežnik	11
Avtentikacija	14
Življenski cikel strežniškega razreda.....	14
REST	14
Odjemalec za Windows/OS X/iOS	15
Dodajanje funkcij.....	16
Odjemalci za mobilne naprave	16
FireDAC.....	17
Viri	19
DataSnap	19
FireDAC.....	20

Vsi programi, omenjeni v tem dokumentu, so na voljo na naslovu

<http://17slon.com/EA/EA-DatabaseXE4.zip>.

Uvod

V svoji dolgi zgodovini se je Delphi uveljavil predvsem kot orodje za hiter razvoj zmogljivih programov za prikaz in obdelavo podatkov, shranjenih v kateri od strežniških podatkovnih baz. Kljub trenutnim »modnim« smernicam, ki večino razvoja postavljajo na lahko odjemalsko platformo (pametni telefoni in tablice), se podpora bazam podatkov iz različice v različico izboljšuje. Na današnjem predavanju si bomo ogledali novosti, ki jih je ogrodje DataSnap dobilo v zadnjih nekaj letih (od RAD Studia 2009 dalje) ter novo infrastrukturo za dostop do baz podatkov FireDAC (na voljo od RAD Studia XE3).

DataSnap

Zgodovina

DataSnap se je v Delphiju pojavil že v pradavni različici Delphi 3, ko se je pojavil kot knjižnica MIDAS, ki je omogočala razvoj večnivojskih (multi-tier) aplikacij. V večnivojskih aplikacijah odjemalski program ne komunicira neposredno s podatkovno bazo, temveč to vlogo prevzame poslovni strežnik (business server), ki podatkov po potrebi ne samo posreduje, temveč tudi obdeluje (spreminja, agregira ...) ter poskrbi za varnostne mehanizme za dostop do podatkov. Za prenos podatkov je MIDAS uporabljal mehanizem COM, v nekem trenutku pa tudi protokol CORBA, ki je podatke prenašal v enkapsulaciji SOAP.

MIDAS se je v Delphiju 6 preimenoval v DataSnap, same korenine orodja pa so ostale v obliki knjižnice midas.dll, ki je RAD Studio priložene tudi v najnovejši različici XE4. Od različice 2009 dalje je DataSnap zasnovan na podatkovnem stroju dbExpress ter za komunikacijo med odjemalcem in poslovnim strežnikom uporablja vseprisotni internetni protokol TCP/IP, torej običajno internetno povezavo. V različici 2010 je bila dodana še podpora za komunikacijski mehanizem HTTP ter shranjevanje podatkov v zapisu JSON, kar je omogočilo razvoj odjemalcev, ki tečejo v brskalnikih ali na mobilnih napravah ter s strežnikom komunicirajo po standardu REST. O tej različici ogrodja DataSnap bo tekla beseda danes.

DataSnap v RAD Studiu XE4

Ogrodje DataSnap ni podprt v različici Professional, temveč zanj potrebujete vsaj različico Enterprise. Strežniški del aplikacije (poslovni strežnik) mora biti preveden za Windows, izdelamo pa lahko 32- ali 64-bitno različico programa. Pri odjemalcu nismo tako omejeni, saj lahko v samem RAD Studiu izdelamo odjemalce za 32- ali 64-bitne Windows, OS X (samo 32-bit) ter iOS. Strežnike in odjemalce lahko razvijamo v jezikih Delphi in C++ za vmesniške platforme VCL in FireMonkey (samo odjemalci), podprt pa je tudi izvajanje v obliki konzolnih aplikacij, Windows servisov ter v strežniku IIS v obliki knjižnic ISAPI.

Z mobilnimi konektorji lahko izdelamo tudi odjemalce za Android, Blackberry, Windows Phone, brskalnike na namiznih računalnikih in še kaj, le da moramo v tem primeru program razviti v specifičnem orodju in jeziku za platformo, ki jo želimo podpirati (Java, C#, JavaScript ...).

Seveda lahko izdelamo tudi več odjemalcev za različne platforme, saj je ravno v tem bistvo večnivojskega dela – poslovno logiko skrijemo v strežnik, ki je en sam, za posamezne platforme pa napišemo le bolj ali manj »lahke« odjemalce.

Rabo ogrodja DataSnap si bomo ogledali skozi nekaj primerov.

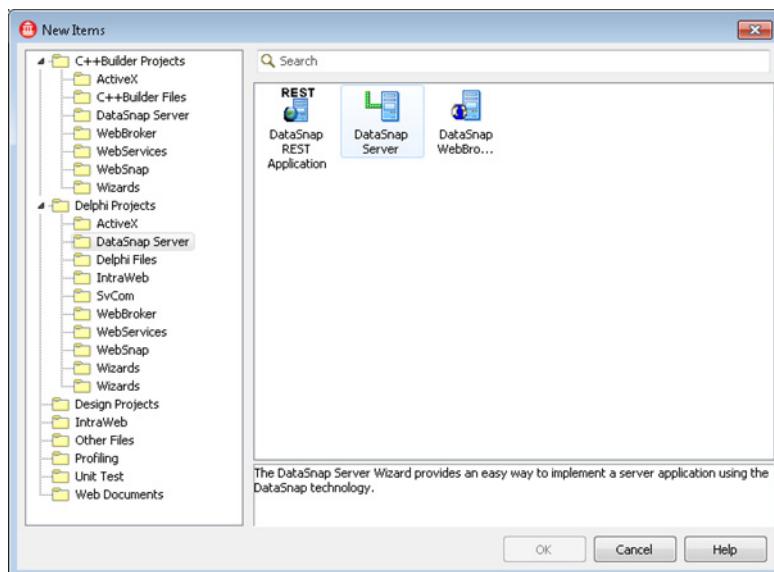
Večnivojski dostop do baze podatkov

Izdelavo večnivojskega programa je v večini primerov najbolje začeti s strežniškim delom. S tem imamo pri razvoju odjemalca na voljo dostop do podatkov in lahko obrazce oblikujemo z živimi podatki. Strežnik lahko naredimo »ročno« (začnemo nov projekt in nanj postavimo ustrezne komponente), dosti manj dela pa bomo imeli, če uporabimo čarownik.

Končna različica programa je shranjena v mapi SimpleDSMultiTier.

Strežnik

Poženemo RAD Studio ter kliknemo *File, New in Other*. V veji ustreznega programskega jezika (*Delphi Projects* ali *C++Builder Projects*) poiščemo podvejo *DataSnap Server*. Vsi primeri v tej skripti so prikazani na primeru jezika Delphi, enakovredno pa bi jih lahko izdelali za C++ Builder.

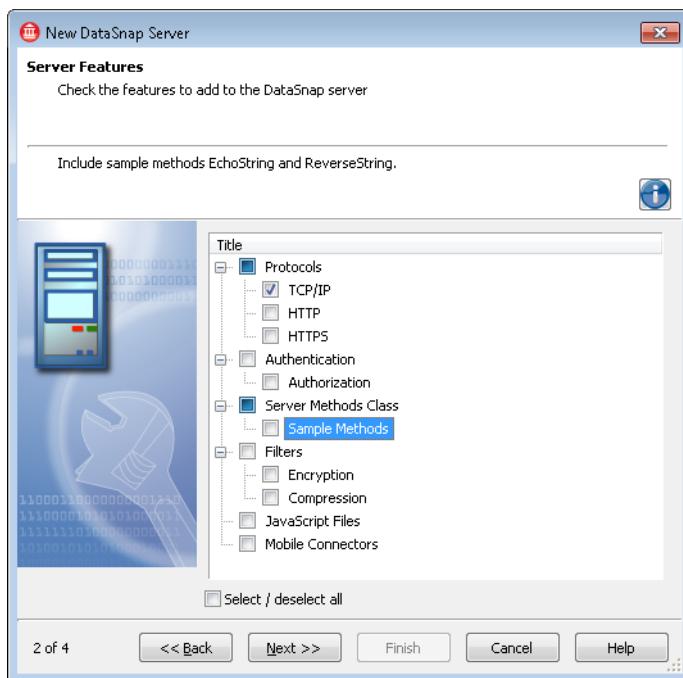


Na voljo so nam trije čarowniki. *DataSnap WebBroker Application* izdela ogrodje za strežnik DataSnap, ki z odjemalci komunicira »po starem«, skozi COM, ter je primeren za vgradnjo v strežnik IIS. Čarownik *DataSnap Server* izdela strežnik s komunikacijo TCP/IP, čarownik *DataSnap REST Application* pa strežnik, ki komunicira po standardih HTTP, REST in JSON in je s tem primeren za izdelavo mobilnih odjemalcev v okoljih, za katere programov ne moremo izdelati v RAD Studiu.

V tem primeru bomo uporabili čarownik *DataSnap Server*. Izberemo ga in kliknemo *OK*. Na naslednji strani čarownika izberemo vrsto aplikacije. Pri temu tipu strežnika so nam na voljo aplikacija VCL, konzolna aplikacija in Windows servis. Izdelali bomo kar običajno aplikacijo VCL. Z nekaj telovadbe lahko izdelamo tudi programsko skupino z vsemi tremi tipi aplikacij, ki uporabljajo isto strežniško kodo. V povezavah na koncu skripte boste našli napotke za to.

Na tretji strani čarownika podrobno nastavimo, kakšno funkcionalnost potrebujemo. Med protokoli pustimo prižgan le TCP/IP (drugi dve možnosti potrebujemo v strežnikih REST). Avtentifikacija (prijava uporabnikov) ter avtorizacija (nastavitev pravic na nivoju objektov) naj zaenkrat ostaneta ugasnjeni, saj ju je zelo enostavno dodati kasneje. V veji *Server Method Class* ugasnemo le okence *Sample Methods* – ta razred nam bo kasneje prišel prav, zato naj ga čarownik le naredi, primerov metod pa ne potrebujemo.

Ostala okenca pustimo prazna, z njimi se bomo ukvarjali kasneje.



Na tretji strani moramo vnesti vrata TCP (port) na katerih bo poslušal strežnik. Privzeta vrednost 211 je primerna za testiranje, pri postavitvi strežnika pa raje izberemo vrata s številko nad 1024 ter to možnost naredimo nastavljivo, saj obstaja možnost, da bodo na pravem strežniku izbrana vrata že zasedena. Gumb *Test Port* preveri, da na računalniku ni nobenega programa, ki bi poslušal na teh vratih, gumb *Find Open Port* pa najde prosta vrata.



Za konec se moramo odločiti, katera komponenta bo uporabljena kot vsebnik (container) naših strežniških razredov. Priporočamo, da vedno uporabite *TDSServerModule*, o razliki med možnostmi pa se poučite v dokumentu *The New DataSnap in Delphi 2009* (povezavo najdete na koncu skripte).

Čarovnik izdela prazen glavni obrazec, prazno enoto *ServerMethodsUnit1*, izpeljano iz *TDSServerModule* ter podatkovni modul (data module) *ServerContainerUnit1*. Slednji vsebuje tri komponente, nujno potrebovane za delovanje strežnika DataSnap.

TDSServer je glavna strežniška komponenta. Njena najpomembnejša lastnost, *AutoStart*, je že privzeto vključena – to pomeni, da se bo strežnik samodejno zagnal.

TDSServerClass določa *razred*, skupek funkcionalnosti, ki jo nudimo odjemalcem. Komponenta povezuje strežnik (*DSServer1*) z razredom, ki implementira funkcionalnost (*TServerMethods1* v enoti *ServerMethodsUnit1*). Povezava je narejena tako, da lastnost *Server* kaže na instanco serverja (*DSServer1*), v kodi pa sprogramiramo dogodek *OnGetClass*, ki vrne ime razreda (*TServerMethods1*). To ime bomo kasneje potrebovali v odjemalski kodi.

DSServer1 TDSServer

Properties	
AutoStart	<input checked="" type="checkbox"/> True
ChannelQueueSize	100
ChannelResponseT	30000
LiveBindings Design	
Name	DSServer1
Tag	0

DSServerClass1 TDSServerClass

Properties	
LifeCycle	Session
LiveBindings Design	
Name	DSServerClass1
Server	DSServer1
Tag	0

```

procedure TServerContainer1.DSServerClass1GetClass(
  DSServerClass: TDSServerClass; var PersistentClass: TPersistentClass);
begin
  PersistentClass := ServerMethodsUnit1.TServerMethods1;
end;

```

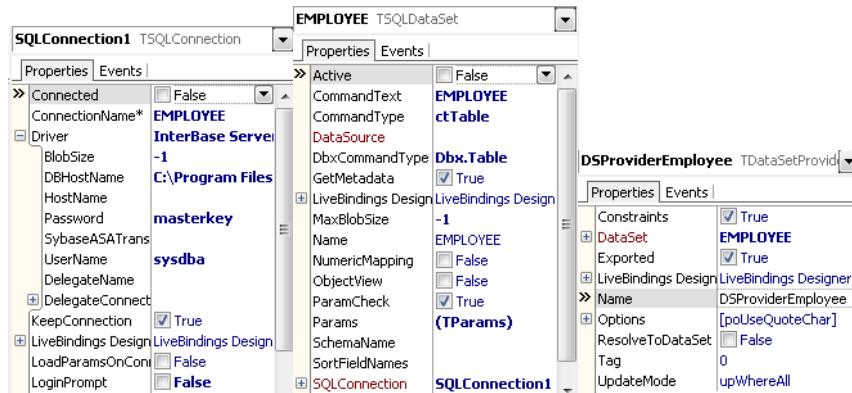
TDSTCPServerTransport določa lastnosti komunikacijskega kanala. Tu lahko nastavimo največje število obdelovalnih niti (*MaxThreads*) in vrata (*Port*), seveda pa moramo komponento tudi povezati s strežnikom (*Server*).



V tem trenutku lahko strežnik že poženemo (F9). Če imate aktiven vdorobran (firewall), se bo najverjetneje pojavilo še vprašanje, če želite aplikaciji dovoliti komunikacijo po privatnih in javnih omrežjih. Prvo možnost vsekakor odobrite, druge pa najverjetneje ne boste potrebovali. Kaj posebnega še ne moremo videti – program se požene s praznim glavnim oknom in to je to. Ugasnimo ga in nastavimo komunikacijo s podatkovnim strežnikom.

Na modul *ServerMethodsUnit1* naložimo:

- *TSQLConnection*, povezan s podatkovnim strežnikom. V našem primeru smo uporabili kar InterBase in priloženo bazo *Employee.GDB*.
- *TSQLDataSet*, povezan na *TSQLConnection*. Nastavimo ga tako, da vrača tabelo *EMPLOYEE* (*CommandType* := *ctTable*, *CommandText* := *EMPLOYEE*, *Name* := *EMPLOYEE*).
- *TDataSetProvider*, povezan na *TSQLDataSet*. Nastavimo mu razumljivo ime (*DSProviderEmployee*), ker ga bomo potrebovali v odjemalcu.



Med nastavljanjem preizkusimo pravilnost nastavitev, tako da komponento aktiviramo (*Connected* := *True*, *Active* := *True*), preden projekt shranimo, pa komponente deaktiviramo.

Na koncu projekt poženemo brez razhroščevalnika (Ctrl+Shift+F9). Pognani strežnik potrebujemo za testiranje odjemalca.

Odjemalec

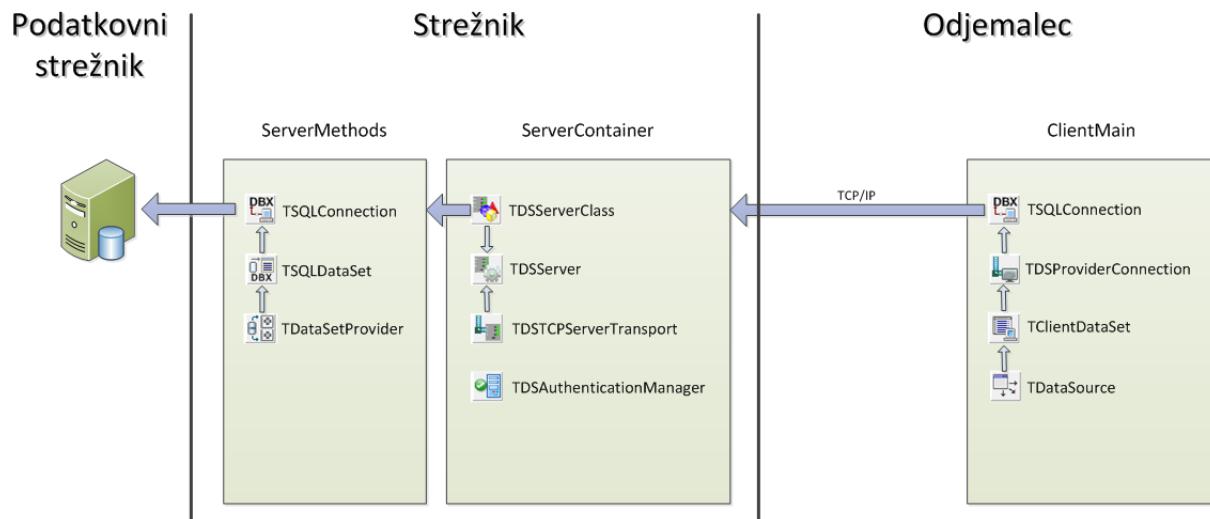
Tudi odjemalca lahko naredimo s čarovnikom, ker pa je delo tu bolj enostavno, lahko našega prvega odjemalca naredimo tudi »ročno«.

Najprej izdelamo novo aplikacijo VCL (lahko pa bi uporabili tudi FireMonkey). Nanjo postavimo naslednje komponente:

- *TSQLConnection*. S to komponento bomo vzpostavili povezavo s poslovnim strežnikom. Lastnost *Driver* nastavimo na *DataSnap* ter ugasnemo pozivnik za uporabniškim imenom in geslom (*LoginPrompt*). Znotraj lastnosti *Driver* sta ime in vrata ciljnega strežnika že pravilno nastavljena (*HostName = 'localhost'*, *Port = 211*), v praksi pa bo treba ta podatek spremeniti, ker odjemalec in strežnik tipično ne bosta pogana na istem računalniku, vrata pa ne bodo 211. Ker je strežnik pognan, lahko delovanje takoj preizkusimo (*Connected := True*).
- *TDSProviderConnection*. To je »dataset provider« za DataSnap. Nastaviti moramo lastnost *SQLConnection* (na pravkar narejeni *TSQLConnection*) ter *ClassName* na ime modula z razredi v strežniku (*TServerMethods1*).
- *TClientDataSet*. Lokalna kopija podatkov. Nastaviti moramo *RemoteServer* na *TDSProviderConnection*, ki smo ga ravno kar naredili, nato pa iz seznama *ProviderName* izberemo edinega implementiranega ponudnika podatkov – *DSPublisherEmployee*.
- Na koncu na obrazec postavimo še običajne komponente za dostop do podatkov – *TDataSource*, *TDBGrid* in *TDBNavigator*. Če je vse pravilno nastavljeno in so ustrezne komponente (*TSQLConnection*, *TDSProviderConnection*, *TClientDataSet*) povezane in aktivirane, bomo v podatkovnem gridu že videli podatke.
- Komponenti *TClientDataSet* nastavimo še dogodek *OnAfterPost* in vanj napišemo: (*DataSet as TClientDataSet*) `.ApplyUpdates(0)`; Zdaj bomo lahko tabelo tudi urejali.

Program le še poženemo in moral bi pravilno delovati.

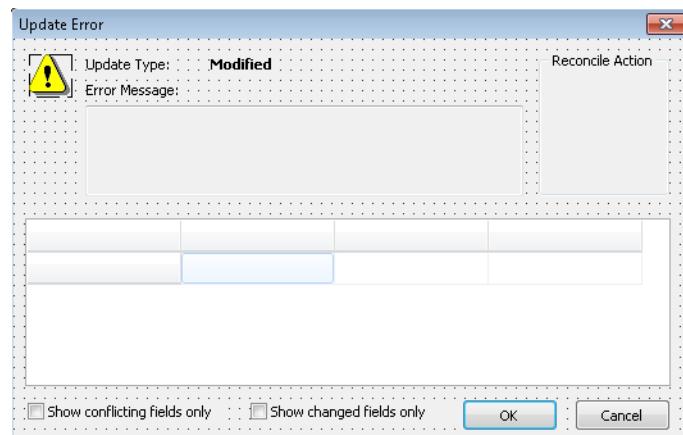
Spodnja slika prikazuje povezave in komunikacijske poti med komponentami.



Obravnavanje napak

Pri hkratnem delu več uporabnikov lahko pri shranjevanju sprememb podatkov pride do konfliktov. V strežniku ali odjemalcu moramo zato nekako vgraditi funkcionalnost, ki bo konflikte razreševala. Izmislimo si lahko lastna pravila, ali pa odločitev prepustimo uporabniku. Ogledali si bomo slednjo možnost.

V odjemalcu potrebujemo nov obrazec, izpeljan iz standardnega obrazca za obravnavanje napak. Dobimo ga z ukazom *New, Delphi Projects, Delphi Files, VCL Reconcile Error Dialog*.

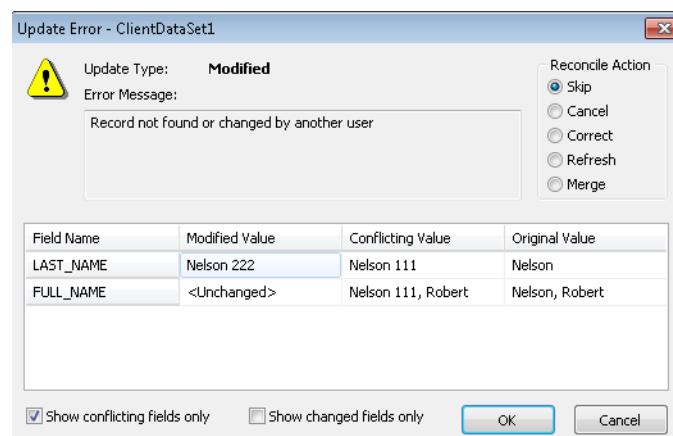


Obliko obrazca si lahko prilagodimo po željah ter ga po potrebi prevedemo. Obdržati pa mora iste sestavne dele. Poskrbimo tudi, da obrazec ni samodejno kreiran ob zagonu programa. Shranimo ga v projekt, nato pa ga v formi, kjer obdelujemo podatke (pri nas je to kar glavna forma), dodamo v seznam *uses*.

Sprogramiramo dogodek *TClientDataSet.OnReconcileError* in vanj dodamo eno vrstico.

```
procedure TForm64.ClientDataSet1ReconcileError(
  DataSet: TCustomClientDataSet;
  E: EReconcileError; UpdateKind: TUpdateKind;
  var Action: TReconcileAction);
begin
  Action := HandleReconcileError(DataSet, UpdateKind, E);
end;
```

Delovanje preizkusimo, tako da poženemo dve kopiji programa ter v njiju poskusimo spremeniti in shraniti isti zapis. Dobili bomo sporočilo o napaki, podobno spodnjemu.



Avtentikacija

Avtentikacijo (overovitev) uporabnikov na podlagi uporabniškega imena in gesla dodamo zelo enostavno. Spremeniti moramo tako strežnik kakor tudi odjemalec, zato oba programa ugasnemo.

V strežniški aplikaciji na modul *ServerContainerUnit1* dodamo komponento *TDSAuthenticationManager* ter ji nastavimo dogodek *OnAuthenticate*, v katerem preverimo uporabniško ime in geslo. (Namig – v skladu z napisanimi standardi je pri preverjanju uporabniškega imena smiselno ignorirati razlike med malimi in velikimi črkami, pri preverjanju gesla pa ne.)

```
procedure TServerContainer1.DSAuthenticationManager1UserAuthenticate(
  Sender: TObject; const Protocol, Context, User, Password: string;
  var valid: Boolean; UserRoles: TStrings);
begin
  valid := SameText(user, 'delphi') and SameStr(password, 'lju');
end;
```

V odjemalcu uporabniško ime in geslo nastavimi znotraj lastnosti *Driver* komponente *TSQLConnection*. Nastaviti moramo lastnosti *DSAuthUser* in *DSAuthPassword*.

	Driver	DataSnap
BufferKBSIZE	32	
CommunicationIP		
CommunicationPort	tcp/ip	
CommunicationTimeout		
ConnectTimeout		
DatasnapContext	datasnap/	
DSAuthPassword	lju	
DSAuthScheme		
DSAuthUser	delphi	

Filtri

Podatke na komunikacijskem kanalu (TCP/IP) lahko spremojmo s *filtri*. Z njimi lahko podatke stisnemo (kompresija), jih podpišemo ter s tem zavarujemo pred spremenjanjem (hash), jih šifiramo (enkripcija), jih zapisujemo v dnevnik in še in še. Primere prvih treh filtrov dobimo z RAD Studiom, imenujejo pa se ZLIB (kompresija), RSA (podpisovanje) in PC1 (šifriranje). Ker je šifrirni algoritem PC1 zelo šibak (ZDA enačijo šifrirne postopke z municijo in zelo omejujejo izvoz), je bolje uporabiti bolj varnega. Med povezavami na koncu skripte boste našli spletno stran, kjer so našteti vsi pomembni filtri za DataSnap, med njimi tudi šifrirni filter, ki podpira zelo varna mehanizma Blowfish in Rijndael.

Na strežniku dodamo filtre, tako da »odpremo« lastnost *Filters* komponente *TDSTCPTransport* ter izberemo enega ali več filtrov. V našem primeru smo dodali filter *ZlibCompression*, ki stisne podatke pred pošiljanjem po omrežju.

Na odjemalcu moramo v seznam *uses* dodati enoto, ki implementira filter. Filter ZLIB je implementiran v enoti *Data.DbxCCompressionFilter*, filtra PC1 in RSA pa v *Data.DbxSocketChannelNative*.

Na obeh straneh (v strežniku in odjemalcu) moramo vedno uporabiti iste filtre.

Življenjski cikel strežniškega razreda

V strežniku ne definiramo konkretnega primerka (instance) strežniškega razreda (v našem primeru *TServerMethods1*), temveč le njen razred. Primerek naredi po potrebi strežnik sam, natančno obnašanje pa določa lastnost *LifeCycle* komponente *TDSServerClass*. Lastnost ima lahko sledeče vrednosti:

- *Session*: Nov razred se naredi ob povezavi klienta, instanca pa se uniči, ko se klient odjaví. To je privzeto obnašanje. V tem načinu lahko v strežniškem razredu definiramo polja, ki bodo lastna vsaki povezavi, ter v njih shranimo določeno stanje (stanje seje).

- *Invocation*: Nov razred se naredi ob izvedbi vsakega ukaza. Stanja seje v strežniškem razredu zato ne moremo hrani. Pravimo, da strežnik deluje brez notranjega stanja (stateless).
- *Server*: Vse povezave si delijo isti primerek razreda. Če uporabimo ta način, moramo izredno paziti, da vse komponente in metode razreda delujejo pravilno, kadar jih uporabljam iz več niti hkrati (thread-safe).

Program *SimpleDSMultiTier* vsebuje gumb *Plus 1*, s katerim izvedemo lastno metodo *PlusOne* na strežniku. Z njim lahko primerjamo delovanje strežnika v vseh treh primerih. Kako izdelamo in uporabimo lastno metodo, si bomo ogledali v primeru *DataSnap in REST*.

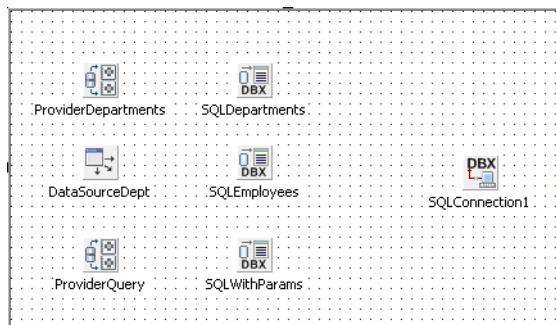
Poizvedbe s parametri

Odjemalec, ki vedno zahteva celo tabelo, ne spada ravno med »lahke« in ne prepušča dela poslovnemu strežniku. Zato si oglejmo še, kako na poslovnem strežniku napišemo ponudnik podatkov (provider), ki vrne rezultat parameterizirane poizvedbe SQL.

Program je shranjen v mapi DSPParamQuery.

Strežnik

V mapi *DSPParamQuery* najdemo dva programa – strežnik *DSPParamQueryServer* in odjemalec *DSPParamQueryClient*. Na modulu *ServerMethods* je ena povezava s podatkovno bazo (*SQLConnection1*) ter dva ponudnika podatkov – *ProviderDepartments* in *ProviderQuery*.



Ponudnik *ProviderDepartments* je povezan na *TSQLDataSet SQLDepartments*, ki vrača rezultate SQL poizvedbe »select * from DEPARTMENTS«. Ponudnik *ProviderQuery* pa je povezan na *SQLWithParams*, ki je *TSQLDataSet* s poizvedbo »select * from EMPLOYEE where job_code = :job_code« in parametrom *job_code*. Tej poizvedbi bo ustrezhen parameter poslat odjemalec.

Odjemalec

Na odjemalcu imamo en *TSQLEConnection*, povezan na strežnik DataSnap, kot smo se naučili v prejšnjem primeru, ter en *TDSProviderConnection*, priključen na strežniški razred *TServerMethods1* (spet tako, kot v prejšnjem primeru). Na obrazcu sta še dve komponenti *TClientDataSet*, vsaka z ustrezačim *TDataSource* in vizualno komponento *TDBGrid*. Na obrazcu je še izbirnik *TcomboBox*, s katerim bomo nastavili parameter za poizvedbo.

Prvi *TClientDataSet* ima nastavljen *ProviderName* na *ProviderDepartments*. S tem dobi od strežnika (skozi ponudnik *ProviderDepartments*) celo tabelo DEPARTMENTS. Drugi *TClientDataSet* ima

nastavljen *ProviderName* na *ProviderQuery* in ima nastavljen en parameter. Ime parametra ni pomembno, le tip parametra mora ustrezati tipu na strežniku.

Ko spremenimo izbirnik *TComboBox*, se izvede naslednja koda:

```
procedure TfrmDSPParamQueryClient.ComboBox1Change(Sender: TObject);
begin
  cdsQuery.Close;
  cdsQuery.Params[0].AsString := ComboBox1.Text;
  cdsQuery.Open;
end;
```

Koda zapre *TClientDataSet*, nastavi parameter, ter ponovno odpre *TClientDataSet*. S tem dobimo v drugi *TDBGrid* ustrezno filtrirane podatke.

The screenshot shows the 'DataSnap Parameterized Query' dialog box. It contains two *TDBGrid* components. The top grid has columns: DEPT_NO, DEPARTMENT, HEAD_DEPT, MNGR_NO, and BUDGET. The bottom grid has columns: EMP_NO, FIRST_NAME, LAST_NAME, PHONE_EXT, and HIRE_DATE. Both grids show data from the Northwind database.

DEPT_NO	DEPARTMENT	HEAD_DEPT	MNGR_NO	BUDGET
000	Corporate Headquarters		105	100000
100	Sales and Marketing	000	85	200000
110	Pacific Rim Headquarters	100	34	60000
115	Field Office: Japan	110	118	50000
116	Field Office: Singapore	110		30000
120	European Headquarters	100	36	70000
121	Field Office: Switzerland	120	141	50000
123	Field Office: France	120	134	40000

EMP_NO	FIRST_NAME	LAST_NAME	PHONE_EXT	HIRE_DATE
9	Phil	Forest	229	17.4.1989
15	Katherine	Young	231	14.6.1990
20	Chris	Papadopoulos	887	1.1.1990
94	Randy	Williams	892	8.8.1992

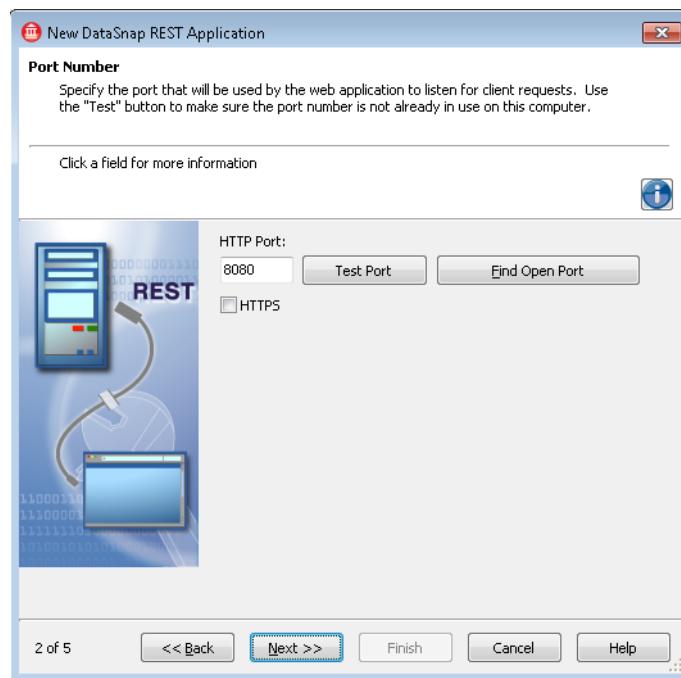
DataSnap in REST

Za konec si bomo ogledali še izdelavo strežnika s tehnologijo REST ter izdelavo različnih vrst odjemalskih programov zanj. Tokrat bomo uporabili priložena čarovnika za izdelavo tako strežnika kakor tudi odjemalca, napisanega v Delphiju.

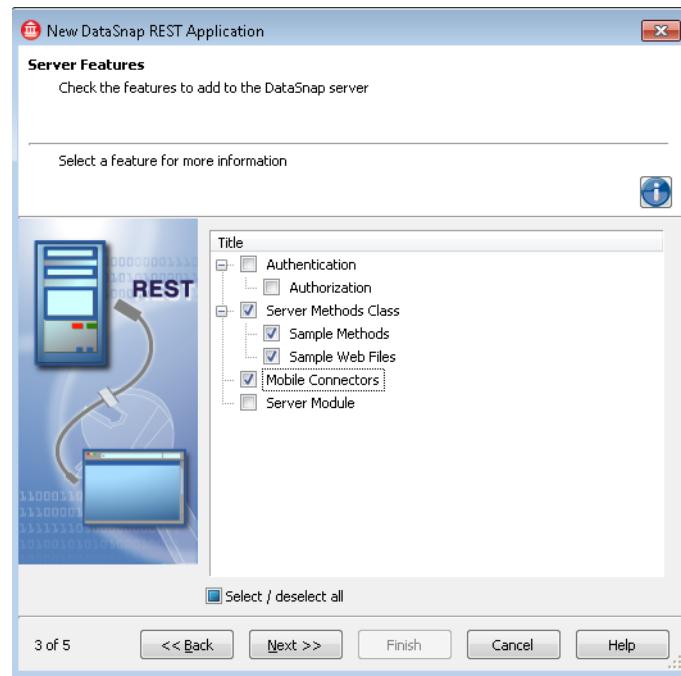
Program je shranjen v mapi DataSnapREST.

Strežnik

Za izdelavo strežnika uporabimo čarovnik *DataSnap REST Application*. Ta nam nudi drugačne možnosti kakor prej uporabljeni čarovnik: aplikacijo VCL, konzolno aplikacijo in dinamično knjižnico ISAPI, ki jo lahko integriramo v strežnik IIS. Podati moramo vrata (tako kot v prejšnjem primeru), na katerih bo v tem primeru poslušal spletni strežnik, ki bo uporabljal protokola HTTP in/ali HTTPS. Če uporabljamo slednjega, moramo nastaviti tudi spletne certifikate, zato bomo v našem enostavnem primeru ta korak izpustili.

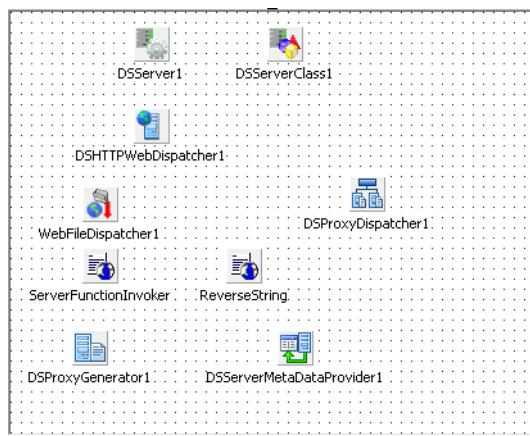


Pri konfiguraciji imamo manj možnosti kakor poprej. Poleg privzetih bomo vključili še možnost *Mobile Connectors*, ki pripravi knjižnice za izdelavo odjemalcev na mobilnih napravah.



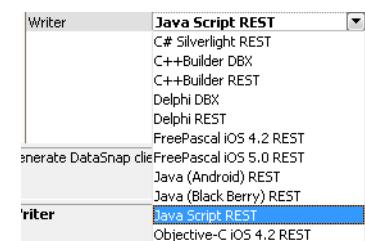
Na koncu moramo zopet določiti vrsto strežniškega modula. Tako kot poprej uporabimo *TDSModule*. Določiti moramo še mapo, v katero se bo shranil projekt.

Čarownik bo naredil projekt, ki vsebuje spletni modul (Web Module), na njem pa je cela množica komponent.



Poleg že znanih *TDSServer* in *TDSServerClass* je tu še množica komponent, ki skrbijo za pravilno vračanje spletnih strani in klicanje funkcij iz strežniškega razreda (*TServerClass1*). Pomembna sta *ServerFunctionInvoker* in *ReverseString* – komponenti, ki kličeta strežniške funkcije in rezultat vstavita v predlogo HTML. Dve sta tu zato, da vidite, da ni nujno, da bi se vse funkcije izvajale po istem kopitu in vračale strani na enak način. Vsebino, ki ju vračata, si lahko ogledate, tako da se iz spletnega brskalnika priključite na <http://localhost:8080> (na vratih 8080 posluša naš strežnik). (Mimogrede, če boste do strežnika dostopali le z odjemalskimi aplikacijami, spletnne predloge in na tak način sestavljeni rezultati niso pomembni. Ves prenos podatkov bo potekal po standardu JSON.)

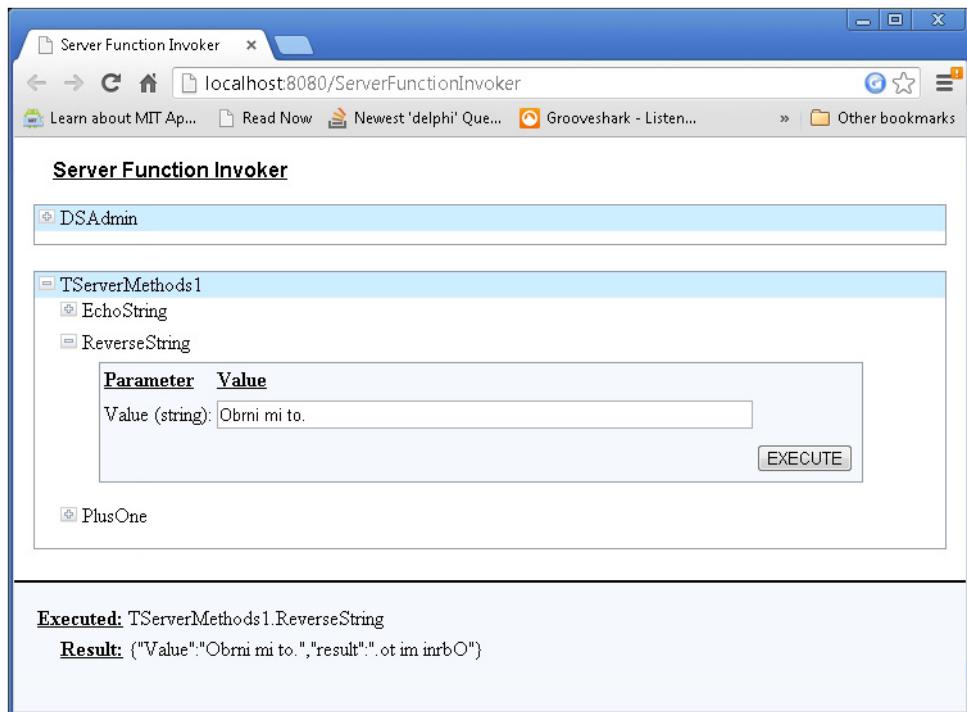
Pomembna je tudi komponenta *TDSProxyGenerator*, ki omogoča izdelavo posredniških (proxy) knjižnic za mobilne naprave. V lastnosti *Writer* si lahko ogledate, katere knjižnice so podprte. Knjižnico za specifično mobilno napravo najlažje izdelamo tako, da poženemo strežnik, ter uporabimo priloženi program *Win32ProxyDownloader*. Postopek je opisan v dokumentaciji; povezavo (*Mobilni konektorji*) najdete na koncu skripte.



Čarownik izdela tudi obsežno strukturo imenikov, od katere je najpomembnejša veja *proxy*, v kateri so shranjene predloge za izdelavo posredniških knjižnic. V veji *templates* so shranjene predloge spletnih strani, v mapi *css* slogovne (oblikovalne) knjižnice, v mapi *images* slike, v mapi *js* pa podporne datoteke JavaScript.



S klikom povezave *Server Functions* pridemo do strani, ki dokumentira vse strežniške funkcije, tako administrativnih (*DSAdmin* – te lahko tudi izklopimo) kakor tudi uporabniške, torej tiste, ki smo jih dodali sami. Te so zbrane v veji z imenom strežniškega razreda (v našem primeru *TServerMethods1*). Če je definiranih več strežniških razredov, bo vidnih več vej. Vsako funkcijo lahko takoj tudi preizkusimo. Ker smo pri izdelavi strežnika izbrali možnost *Sample Methods*, sta nam na voljo testni funkciji *EchoString* in *ReverseString*.



Na primeru na sliki vidimo, da se podatki prenašajo po standardu JSON (vrednost *Result* na dnu zaslona).

Avtentikacija

Overovljanje dodamo na enak način kot v prejšnjih primerih, le da tu uporabimo komponento *TDSHTTPServiceAuthenticationManager*.

Življenski cikel strežniškega razreda

Podobno kakor pri »TCP« strežniku ima razred lastnost *LifeCycle*, ki lahko vsebuje *Session*, *Invocation* in *Server*. REST strežniki pa se razlikujejo po tem, da nikoli ne shranjujejo seje in da vrednost *Session* enak rezultat kakor vrednost *Invocation*. Strežniški razred se vedno izdela na novo ob izvajanju vsakega ukaza.

Nekaj več o sejah in strežnikih DataSnap REST je napisal Marco Cantu v svojem blogu (glej povezavo *Nasveti za izboljšanje hitrosti delovanja*).

REST

Tudi delovanje protokola REST lahko preizkusimo kar v brskalniku. Vedeti moramo le, da so lokacije URL pri upoštevanju pravil REST vedno oblike `http://streznik/datasnap/rest/<strezniskirazred>/<ukaz>/<parametri>`, pri čemer zadnje tri parametre zamenjamo z imenom strežniškega razreda (*TServerMethods1*), imenom ukaza (na primer *ReverseString*) ter parametri. Na tak način bomo sicer lahko poslali le enostavne parametre, a že to zadošča za hitro testiranje.

Če v brskalnik vpišemo <http://localhost:8080/datasnap/rest/TServerMethods1/ReverseString/Delphi>, bomo dobili vrnjen obrnjen niz »Delphi«, seveda shranjen v zapisu JSON.

Z uporabo protokola REST lahko napišemo odjemalca za poljubno platformo brez uporabe kakršnihkoli posredniških knjižnic, res pa imamo lahko s tem kar precej dela.



Odjemalec za Windows/OS X/iOS

Odjemalec za operacijske sisteme, ki jih podpira RAD Studio, najlažje naredimo kar s čarovnikom. Izdelamo novo aplikacijo VCL, nato pa izberemo *File, New, Other*. V veji *DataSnap Server* se pojavijo nove ikone, s katerimi naredimo odjemalske module. Ker uporabljamo strežnik REST, moramo narediti odjemalski modul *DataSnap REST Client Module*. V primeru običajnega strežnika (naši prejšnji primeri) pa bi uporabili *DataSnap Client Module*.

Določiti moramo lokacijo strežnika (krajevni ali oddaljeni; tudi če izberemo slednjega, lahko kasneje za ime nastavimo *localhost* in delamo s krajevnim strežnikom), vrsto strežnika (DataSnap, WebBroker ali IIS) ter naslov, vrata ter uporabniško ime in geslo. Če ne uporabljamo avtorizacije, lahko slednji dve polji pustimo prazni.

Čarovnik bo naredil majhen podatkovni modul, na katerem je le komponenta *TDSRestConnection*, ki skrbi za povezavo s strežnikov, poleg tega pa zelo pomemben razred *ClientClasses*, ki vsebuje »proxy« (posredniške) funkcije z enakimi imeni in tipi parametrov, kot jih definira strežnik. Ko bi radi v odjemalcu uporabili strežniško funkcijo, le pokličemo funkcijo iz tega razreda in podatki se transparentno prenesejo do strežnika, kjer se funkcija izvede, vrne rezultat, ki se prenese nazaj do našega program in vrne kot rezultat posredniške funkcije.

V testnem programu imamo nekaj gumbov, ki kličejo strežniške funkcije. Na njihovem primeru natančno vidimo, kako je treba opraviti tak klic.

```
procedure TfrmDataSnapRESTClient.Button1Click(Sender: TObject);
begin
  Edit2.Text := ClientModule1.ServerMethods1Client.EchoString(Edit1.Text);
end;

procedure TfrmDataSnapRESTClient.Button2Click(Sender: TObject);
begin
  Edit2.Text := ClientModule1.ServerMethods1Client.ReverseString(
    Edit1.Text);
end;
```

Razreda *ClientClasses* nikoli ne urejamo ročno, saj ga je treba ponovno zgenerirati, kadar spremenjamo strežniške funkcije.

Dodajanje funkcij

Na enostavnem primeru si oglejmo, kako v strežnik dodamo novo funkcijo, ki jo bo uporabljal odjemalec. Ker pri uporabi mobilnih odjemalcev ne moremo uporabljati komponent *TClientDataSet*, moramo celotno delovanje, tudi če upravljamo podatkovno bazo, zasnovati okoli lastnih funkcij, ki se na strežniku preslikajo v podatkovne funkcije. Dober primer za enostavno podatkovno aplikacijo, ki podpira REST, je priložen Delphiju (*DataSnap\FishFacts\FishFactsRESTGrp*).

Naš testni program poleg funkcij *EchoString* in *ReverseString*, ki jih je naredil čarovnik, implementira še funkcijo *PlusOne*, ki vrednosti polja prišteje 1, to shrani nazaj v polje in vrne rezultat.

```
type
  TServerMethods1 = class(TDSServerModule)
  private
    FValue: integer;
  public
    function EchoString(Value: string): string;
    function ReverseString(Value: string): string;
    function PlusOne: integer;
  end;

  function TServerMethods1.PlusOne: integer;
begin
  Inc(FValue);
  Result := FValue;
end;
```

S funkcijo *PlusOne* lahko raziskujemo, kako je obnašanje strežnika odvisno od nastavitve lastnosti *LifeCycle*.

Ko spremenimo število ali parametre strežniških funkcij, moramo na novo izdelati tudi posredniške knjižnice odjemalcev – tako navadnih kakor tudi mobilnih. Za slednje smo že povedali, da to naredimo s programom *Win32ProxyDownloader*, v navadnih odjemalcih pa kliknemo z desno tipko na *TDSRestConnection* in izberemo *Generate DataSnap client classes*. Pri tem mora biti strežnik pognan.

V urejevalniku se pojavi nov zavihek z novo vsebino posredniškega razreda. Najbolje je, da vsebino prekopiramo v *ClientClasses* in novi zavihek zavrzem. S tem je že vse pripravljeno za uporabno novih funkcij.

Odjemalci za mobilne naprave

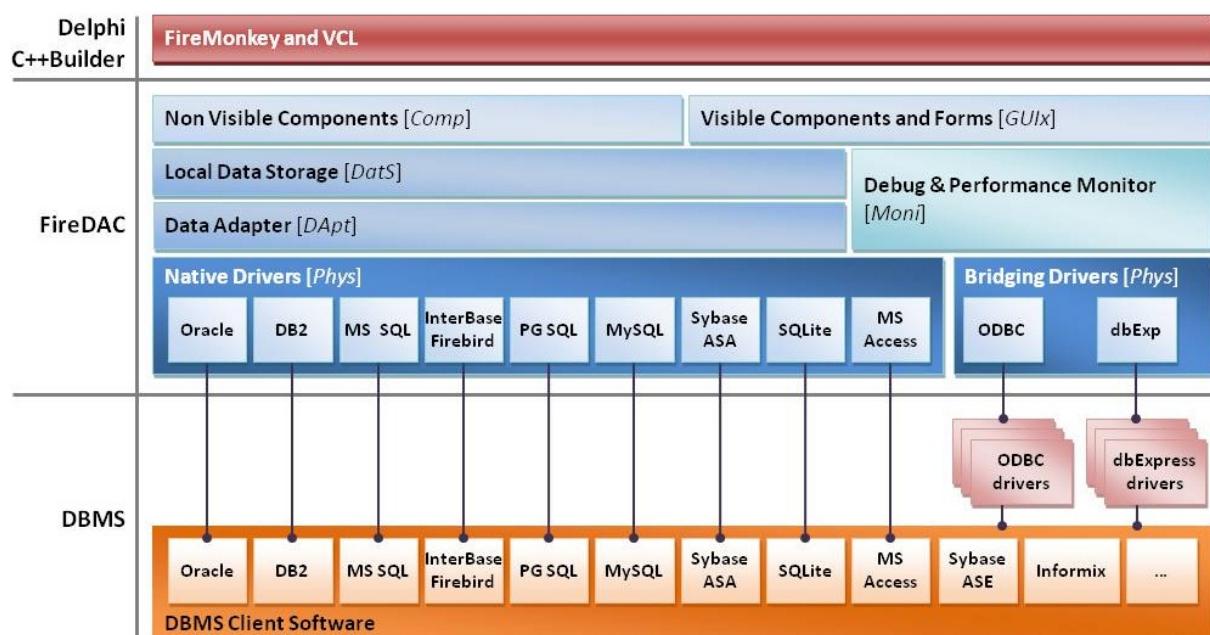
Za nekatere vrste mobilnih naprav lahko odjemalce izdelamo v domorodnih (native) orodjih in za dostop do strežnika uporabimo posredniške knjižnice, ki jih je pripravil *Win32ProxyDownloader*. To lahko storimo na Androidih od 2 do 4, BlackBerryju (knjižnice v Java), Windows Phone 7 (knjižnice v C#), iOS od različice 4 dalje (knjižnice v ObjectiveC), v drugih okoljih pa si lahko pomagamo s posredniškimi knjižnicami v jeziku JavaScript ali pa delamo neposredno s strežnikom z uporabo protokola REST.

FireDAC

FireDAC je nov skupek komponent za dostop do podatkovnih baz. Nov je pravzaprav le po imenu, saj gre za uveljavljene komponente AnyDAC podjetja DASoft, ki jih je Embarcadero kupil in jih sedaj izdaja pod novim imenom. Priložene so RAD Studiu XE4 od različic Enterprise navzgor, lahko pa jih dokupimo tudi za različico Professional. Z malenkostnim trudom lahko FireDAC namestimo tudi v starejše različice RAD Studia in okolja Delphi (povezava *Kako namestiti FireDAC v starejši Delphi*).

Podatkovne komponente so na voljo za okolja Windows, OS X in iOS.

Spodnja slika prikazuje strukturo komponent FireDAC in vse vrste podatkovnih baz, ki jih podpira.



© Embarcadero

Vidimo, da poleg gonilnikov za razne baze dobimo še dva psevdo-gonilnika, ki omogočata rabo poljubnega gonilnika ODBC ali dbExpress. Omogočen je tudi priklop na strežnike DataSnap, čeprav na sliki ni narisani.

Da je bil FireDAC vključen v RAD Studio zelo na hitro, vidimo tudi po primerih, shranjenih v *C:\Program Files (x86)\Embarcadero\FireDAC\Samples*, kjer najdemo podmapo *FPC Specific*. FreePascal je bi res podprt v AnyDACu, v FireDACu pa ne bo več (čeprav trenutna različica z njim še vedno lepo deluje).

Glavni sestavni deli FireDACa so *TADConnection*, *TADQuery*, *TADTable*, *TADTable* in gonilniki, ki jih predstavlja komponente *TADPhysXXXX*. Vsi gonilniki se prevedejo neposredno v aplikacijo. Na voljo je tudi pomnilniška table *TADMemTable* ter množica drugih komponent, od katerih so nekatere splošne (backup & restore, spremljanje delovanja (monitoring) ...), druge pa specifične za določene podatkovne strežnike. Vgrajen je krajevni stroj SQL, s katerim lahko izvajamo poizvedbe SQL na lokalni kopiji podatkov (denimo na *TADMemTable*).

Zanimiva lastnost FireDACA je, da je način dela in razdelitev na komponente zelo podoben BDE-ju. Tako na primer uporaba *TClientDataSet* ni potrebna – *TADQuery* lahko pripnemo neposredno na *TDataSource*. Še vedno lahko uporabimo *TDataSetProvider* (ni pa nujen).

Tabele (*TADTable*) omogočajo hitro dvosmerno navigacijo po velikih naborih podatkov (*Live Data Window*), lahko pa uporabimo tudi enosmerne kurzorje. Shranjevanje spremenjenih podatkov je lahko trenutno ali pa zakasnjeno (cached). Sploh so vse zmožnosti komponent izredno prilagodljive.

Hitro izvajanje velikih količin (skoraj) enakih poizvedb, denimo množice stavkov INSERT ali UPDATE, ki se razlikujejo le po parametrih lahko pospešimo z rabo *Array Data Manipulation Language*, ki take ukaze zapakira v pakete in s tem izredno pospeši obdelavo. Pretvorba v obliko, ki jo razume posamezen podatkovni strežnik, je avtomatska.

Velika prednost FireDAC pred drugimi načini dostopa je, da lahko v ukazih SQL uporabljamo od strežnikov neodvisno sintakso, ki se pred pošiljanjem pretvori v dialect konkretnega strežnika. V posebnih primerih pa lahko v ukazih SQL uporabimo kar pogojno izvajanje ukazov *{if}*, ki omejijo delovanje dela ukaza na posamezno vrsto strežnika. (Primer: »{IF Oracle} SELECT * FROM "REGION"{fi}«.)

Za izdelavo poročil je poskrbljeno z integracijo s programom FastReports.

Viri

DataSnap

»The New DataSnap in Delphi 2009« (članek)

<http://edn.embarcadero.com/article/39227>

»Delphi 2010 DataSnap: Your data – where you want it, how you want it« (članek)

http://www.arena.net.au/embt/edn/Delphi_2010_WP_DataSnap_091016.pdf

»DataSnap in Action« (članek)

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

»Data Access Techniques with ClientDataSets« (članek)

<http://conferences.embarcadero.com/article/32262>

DataSnap REST (dokumentacija)

http://docwiki.embarcadero.com/RADStudio/XE4/en/DataSnap_REST_Messaging_Protocol

Mobilni konektorji (dokumentacija)

http://docwiki.embarcadero.com/RADStudio/XE4/en/Getting_Started_with_DataSnap_Mobile_Connectors

Članki o DataSnapu v reviji Blaise Pascal Magazine (številke #24, #26, #27, #28)

<http://www.blaisepascal.eu>

Kako uporabiti DataSnap v strežnikih Azure

<http://www.uweraabe.de/Blog/2012/09/24/datasnap-in-the-cloud/>

Seznam filtrov za DataSnap

<http://www.danieleteti.it/2009/10/01/datasnap-filters-compendium/>

Napotki za izdelavo Windows servisa, ki vsebuje strežnik DataSnap REST

<http://stackoverflow.com/questions/13066278/datasnap-rest-server-windows-service>

Nasveti za izboljšanje hitrosti delovanja

http://blog.marcocantu.com/blog/datasnap_deployment_performance.html

JSON – JavaScript Object Notation

<http://en.wikipedia.org/wiki/JSON>

REST – REpresentational State Transfer

<http://en.wikipedia.org/wiki/REST>

FireDAC

Domača stran

<http://www.embarcadero.com/products/rad-studio/firedac>

FAQ

<http://www.embarcadero.com/products/rad-studio/firedac-faq>

Dokumentacija

http://docs.embarcadero.com/products/rad_studio/firedac/frames.html

»Introducing FireDAC« (video)

http://www.youtube.com/watch?v=b6Nt_CVKafA

Kako namestiti FireDAC v starejši Delphi

<http://support.embarcadero.com/article/42970>