

A photograph showing a person from behind, sitting at a desk in what appears to be a server room or a technical workspace. They are looking at multiple computer monitors. The background is dimly lit with blue and green lights from the equipment.

EMBARCADERO AKADEMIJA

Embarcadero Akademija 2012

Visual LiveBindings ogrodje za povezovanje z bazami podatkov

Primož Gabrijelčič
<http://www.glagolite.si/training/>

Kazalo

Uvod	2
LiveBindings v Rad Studiu XE2	3
Izrazi	7
Izhodni pretvorniki	9
LiveBindings komponente	9
TBindingsList.....	9
TBindScope in TBindScopeDB.....	10
Povezovalni razredi	10
Povezovanje z objekti	15
LiveBindings in FireMonkey.....	17
Visual LiveBindings v RAD Studiu XE3	19
Čarovnik za povezovanje	19
Vizualno povezovanje.....	20
CustomFormat.....	26
TAdapterBindSource	26
TPrototypeBindSource	29
Ločitev predstavitevnega sloja in poslovnega vmesnika.....	30
Vzorec Model-View-ViewModel (MVVM)	30
Model	31
Model pogleda	31
Pogled.....	31
LiveBindings.....	31
Viri	33
Video.....	34
Dokumentacija	34

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

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

Uvod

V RAD Studiu XE2 je Embarcadero prvič predstavil LiveBindings – ogrodje za prenos podatkov med objekti (ali komponentami) brez programiranja. Z LiveBindings lahko na primer povežemo vsebino gradnika TEdit s poljem v statusni vrstici, seznam objektov z gradnikom TListBox ali pa podatke iz baze »pripnemo« neposredno na preglednico (grid).

LiveBindings (in v XE3 predstavljena nadgradnja Visual LiveBindings) je na voljo v vseh podprtih jezikih (Delphi, C++), v vseh podprtih grafičnih ogrodjih (VCL, FireMonkey) in na vseh podprtih platformah (Windows (32- in 64-bit), OS X in iOS (trenutno le v XE2)).

Uporabniki ogrodja FireMonkey smo ogrodja LiveBindings še posebej veseli zato, ker nam omogoča enostaven prikaz in urejanje podatkov iz podatkovnih baz. Ogrodje FireMonkey namreč ne pozna podatkovno osveščenih (data-aware) gradnikov.

Verjetno pa je še pomembnejša lastnost LiveBindings, da omogoči enostavno ločitev predstavitevenega sloja (forme) od poslovne logike. Tak pristop k programiraju nam olajša preizkušanje programa ter omogoči s kar najmanj truda napisati različne uporabniške vmesnike, ki uporablajo skupno poslovno logiko. To bo še posebej prišlo do izraza v naslednjih letih, ko bomo v RAD Studiu programirali za različne mobilne platforme. Takšnemu pristopu k programiraju smo posvetili nekaj strani na koncu knjige (poglavlje *Ločitev predstavitevenega sloja in poslovnega vmesnika*).

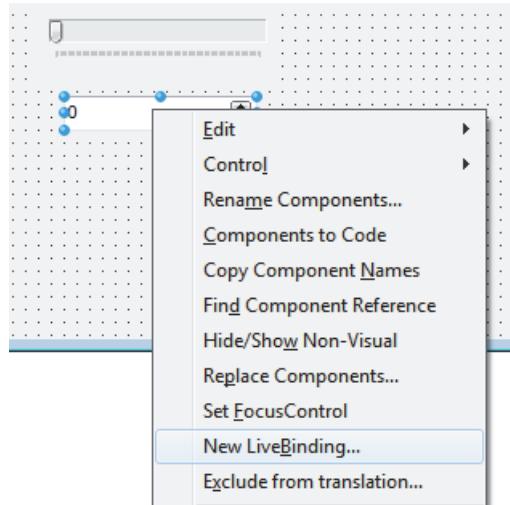
LiveBindings ima tudi nekaj slabosti. Predvsem moramo omeniti zapletenost, slabo pomoč pri iskanju programerjevih napak ter relativno počasnost. Ogrodje namreč temelji na *izrazih* (*expressions*), ki so pravzaprav enostavni fragmenti programa, ki se ne izvajajo v prevedeni obliki, temveč za njihovo izvajanje skrbi tolmač (interpreter). Vseeno pa v večini primerov te razlike v hitrosti ni opaziti. Če pa bi do upočasnitve programa le prišlo, se zavedajte, da vas LiveBindings v ničemer ne ovira pri uporabi klasičnih programerskih pristopov. Vedno lahko uporabite LiveBindings za 99% programa, preostanek pa spišete na klasičen način in s tem pridobite ustrezno hitrost.

V tej knjižici si bomo ogledali najpomembnejše sestavne dele ogrodja LiveBindings in njihovo rabo. Opisani so kronološko, saj novejša različica ogrodja v RAD Studiu XE3 le nadgrajuje osnovo, ki jo je prinesel RAD Studio XE2.

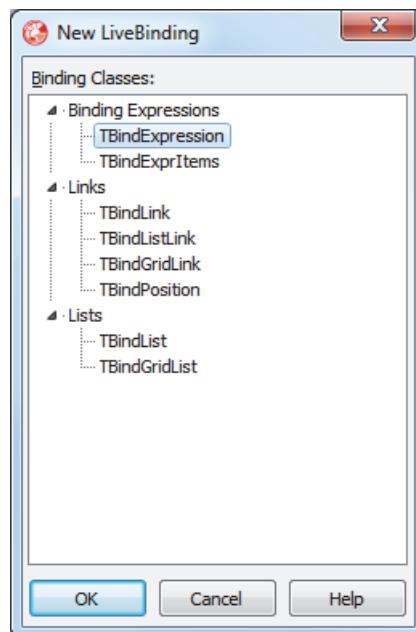
Še opomba: Čeprav vedno omenjamo »RAD Studio XE2« in »RAD Studio XE3«, deluje (Visual) LiveBindings brez omejitev tudi v jezikovno specifičnih različicah, torej v okoljih Delphi XE2/XE3 in C++Builder XE2/XE3.

LiveBindings v Rad Studiu XE2

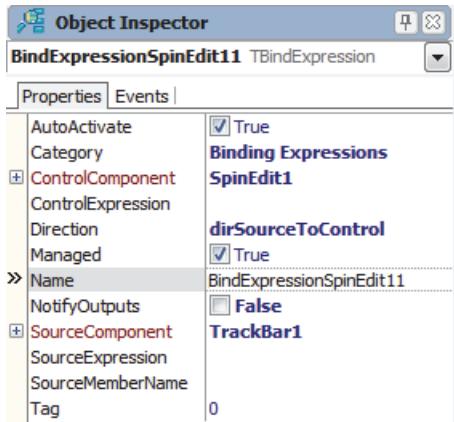
Začnimo z enostavnim primerom (projekt *XE2SimpleBindings*). Naredimo novo aplikacijo ter nanjo odložimo dva gradnika – TTrackBar in TSpinEdit. Gradnik TSpinEdit kliknemo z desno tipko in iz menuja izberemo *New LiveBinding...*



Pojavi se okno *New LiveBinding* v katerem izberemo vrsto povezave. Izberimo *TBindExpression*.

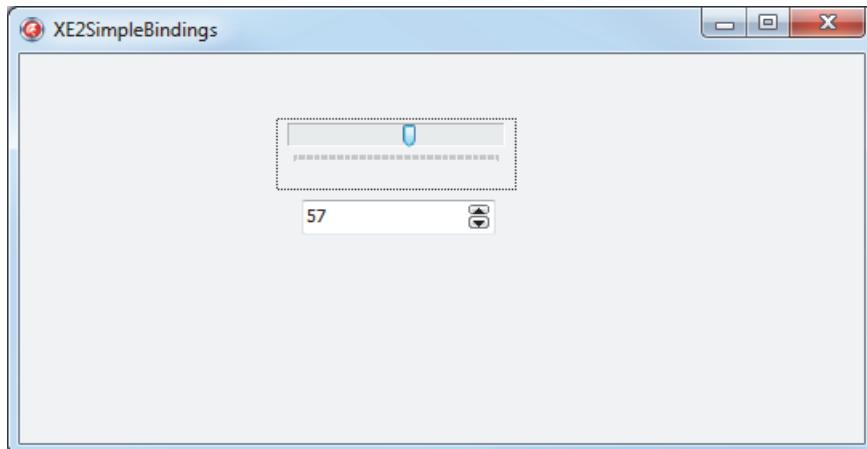


Na obrazcu se pojavi nova komponenta *TBindingsList1*. Če odpremo padajoči izbirnik v oknu *Object Inspector*, pa najdemo še eno novost – nevidno komponento *BindExpressionSpinEdit11*, ki jo v ogrodju LiveBindings imenujemo *živa povezava* (live binding). Ta živa povezava ima že nastavljen ciljni gradnik (ControlComponent) SpinEdit1, nima pa še nastavljenega vira podatkov. Kliknimo izbirnik poleg lastnosti *SourceComponent* in izberimo gradnik TrackBar1.



S tem smo določili objekt, ki predstavlja vir podatkov (TrackBar1) ter objekt, ki bo cilj podatkov (SpinEdit1). Nismo pa še določili iz katere lastnosti izvornega objekta naj LiveBindings bere podatke in kam naj jih zapisuje. Prvo določa lastnost SourceExpression, drugo pa lastnost ControlExpression. Lastnosti sta tipa *string*, zato njunih vrednosti ne moremo izbrati iz seznama, temveč jih moramo vpisati. Prvo nastavimo na Position (vir podatkov bo TrackBar1.Position), drugo pa na Value (podatki se bodo zapisovali v SpinEdit1.Value).

Program poženimo in premaknimo ročico v gradniku TrackBar1. Vidimo, da se spremeni tudi vrednost v vnosnem polju SpinEdit1. Povezali smo dva gradnika, pri tem pa nismo napisali niti vrstice kode!



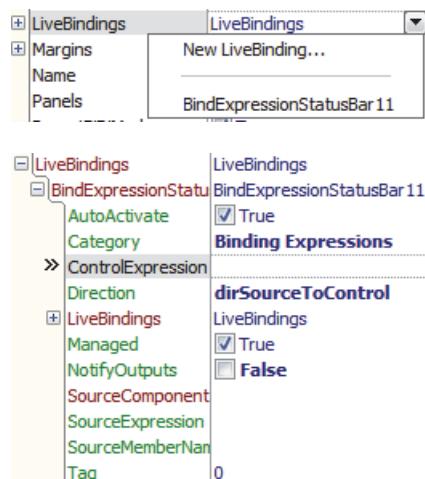
Živo povezavo lahko spremenimo, tako da prenaša podatke v nasprotno smer (od SpinEdit1.Value do TrackBar1.Position) in tudi tako, da prenaša podatke v obe smeri. Smer toka podatkov določa lastnost BindExpressionSpinEdit1.Direction, ki je privzeto nastavljena na dirSourceToControl, dovoljeni vrednosti pa sta še dirControlToSource in dirBidirectional. Če spremenimo lastnost v dirBidirectional in poženemo program, vidimo, da lahko spremojamo vrednost poljubnemu od obenh gradnikov, ogrodje LiveBindings pa bo poskrbelo, da se bo spremenila vsebina povezanega gradnika.

LiveBindings pa ni omejen le na prirejanje vrednosti dveh lastnosti. V lastnost ControlExpression lahko vpišemo izraz, ki uporablja različne funkcije. Zavedati se moramo, da se vrednost tega izraza izračuna šele med izvajanjem programa, za to pa ne poskrbi Delphijev (ali C++Builderjev) prevajalnik, temveč bistveno bolj omejen tolmač (interpreter) ogrodja LiveBindings. Več o tem bomo povedali v nadaljevanju.

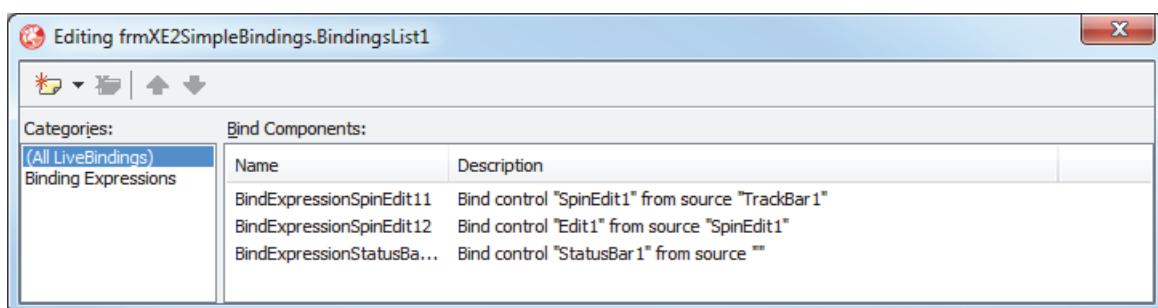
Za demonstracijo postavimo na obrazec še statusno vrstico (TStatusBar) in ji nastavimo lastnost SimplePanel na True. Tokrat uporabimo drugačen način dodajanja žive povezave. (Lahko bi uporabili tudi prej opisani način z desnim klikom, saj sta oba načina enakovredna.) Ko je gradnik StatusBar1 izbran, v oknu Object Inspector poiščemo lastnost LiveBindings in kliknemo puščico desno od nje. Prikazal se bo menu z eno izbiro – *New LiveBinding...*. S klikom dodamo novo živo povezavo BindExpressionStatusBar11.



V predhodnem primeru smo povezavo izbrali v oknu Object Inspector ter jo nato urejali, zdaj pa omenimo še dve možnosti. Prvi način je, da izberemo gradnik StatusBar1 ter v Object Inspectorju kliknemo puščico desno od lastnosti LiveBindings. V menuju imamo sedaj dve možnosti; poleg vedno prisotne *New LiveBinding...* se je pojavila še vrstica *BindExpressionStatusBar11*. S klikom odpremo podvejo nastavitev LiveBinding, kjer lahko nastavimo lastnosti žive povezave. (Prav isto bi dosegli s klikom sličice z znakom + levo od lastnosti LiveBindings.)



Drugi alternativni način je, da dvakrat kliknemo komponento BindingsList1. Pokazalo se bo okno z vsemi živimi povezavami.

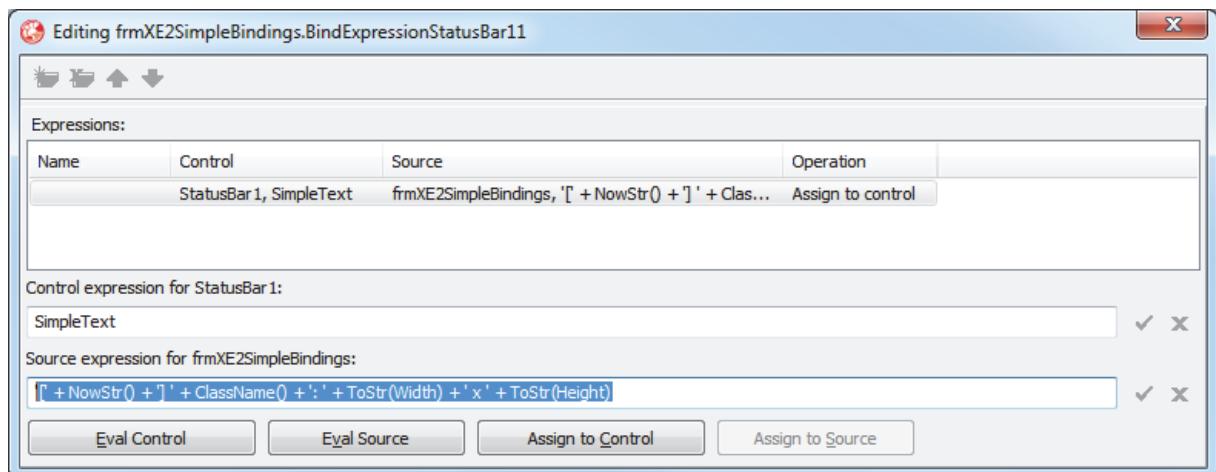


Povezavo lahko kliknemo in jo s tem izberemo v Object Inspectorju ali pa jo dvokliknemo in pridemo do okna, v katerem imamo poleg nastavljanja izvornega in ciljnega izraza (SourceExpression in ControlExpression) tudi možnost izvajanje teh izrazov. Ne moremo pa na tem mestu nastaviti izvornega ali ciljnega objekta (SourceComponent in ControlComponent).

Obe okni sta »plavajoči« - kljub temu, da sta na zaslonu, lahko še vedno normalo delamo z RAD Studiom – urejamo kodo, prevajamo itd. Lahko ju odmaknemo na rob zaslona ali na drugi zaslon, kar pride še posebej prav pri daljšem prilagajanju in preizkušanju zapletenih živih povezav.

Kakor koli se že bomo tega lotili, povezavi BindExpressionStatusBar11 moramo nastaviti naslednje vrednosti:

- SourceComponent = frmXE2SimpleBindings (ime obrazca v testnem programu)
- SourceExpression = '[' + NowStr() + ']' + ClassName() + ':' +ToStr(Width) + 'x' + ToStr(Height)
- ControlExpression = SimpleText

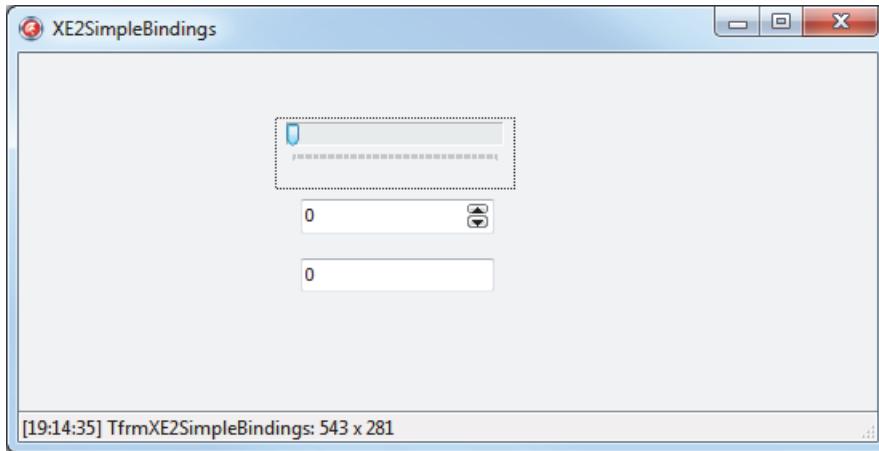


Na klasičen način bi lahko to kodo zapisali kot metodo, pripeto na dogodek OnResize:

```
procedure TfrmXE2SimpleBindings.FormResize(Sender: TObject);
begin
  StatusBar1.SimpleText := '[' + NowStr + ']' + ClassName + ':' + 
    IntToStr(Width) + 'x' + IntToStr(Height);
end;
```

Pozoren bralec je opazil, da izraz ni popolnoma enak tistemu v zgornji sliki. V živi povezavi je vsako ime funkcije zaključeno s parom oklepajev, namesto funkcije IntToStr pa je uporabljena funkcija ToStr. Gre za omejitve tolmača LiveBindings, o katerem bomo še govorili.

Če sedaj izbrišemo dogodek FormResize in program poženemo, vidimo, da se v statusni vrstici pojavi sporočilo z uro, imenom razreda obrazca ter velikostjo obrazca. Če pa obrazcu spremeni velikost, se sporočilo ne spremeni. V prvem primeru (povezava TTrackBar in TSpinEdit) je bila spremembra trenutna. Zakaj?



Razlika nastane zaradi dejstva, da pozna LiveBindings dve vrsti živih povezav – *managed* in *unmanaged*. Druge se posodobijo samodejno (da, ime je zavajajoče), pri prvih pa moramo posodobitev sprožiti ročno. Le redki gradniki podpirajo samodejno osveževanje in obrazec ne spada med njih. Zato moram posodobitev sprožiti ročno, tako da spišemo enostavno metodo dogodka OnResize.

```
procedure TfrmXE2SimpleBindings.FormResize(Sender: TObject);
begin
  BindingsList1.Notify(Sender, '');
end;
```

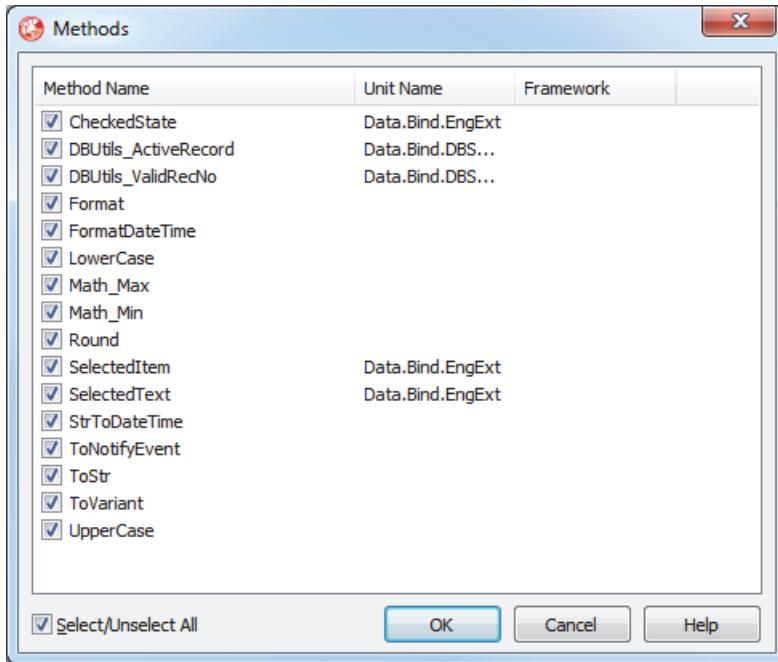
Klic metode Notify sporoči komponenti BindingsList1, da se je spremenila vrednost objekta (Sender, v tem primeru sam obrazec, torej objekt frmXE2SimpleBindings). Kot drugi parameter bi lahko poslali lastnosti, ki se je spremenila. Ker smo poslali prazen niz, bo LiveBindings pregledal vse žive povezave, povezane z obrazcem in priredil ciljnim objektom na novo izračunane vrednosti.

Izrazi

Omenili smo že, da je v ogrodje LiveBindings vgrajen tolmač, ki zna izračunati vrednost izrazov iz nekega omejenega, Pascalu podobnega jezika. Izrazi lahko vsebujejo konstante (nizi, števila), pri tem pa so nizi lahko omejeni z enojnimi ('') ali dvojnimi ("") narekovaji. Uporabimo lahko operatorje (+ - * / < >) ter oklepaje za vrstni red določanja izrazov.

Izrazi lahko dostopajo do lastnosti objektov, nad katerimi deluje živa povezava (SourceComponent, ControlComponent), prav tako pa do lastnosti objektov, iz katerih deduje izvorni ali ciljni objekt. Povedano drugače, če »priplnete« živo povezavo na vizualni gradnik, so vam na voljo tudi lastnosti razreda TComponent, TPersistent, TObject itd. Ker je del vsakega vizualnega gradnika tudi lastnost Owner, ki vsebuje obrazec (formo), lahko vsak izraz, ki izvira iz grafičnega gradnika (kontrole), dostopa tudi do lastnosti same forme.

Uporabimo lahko tudi nekaj vgrajenih funkcij. Seznam podprtih funkcij si ogledamo, tako da izberemo gradnik BindingsList1 in v Object Inspectorju kliknemo tri pikice desno od lastnosti Methods.



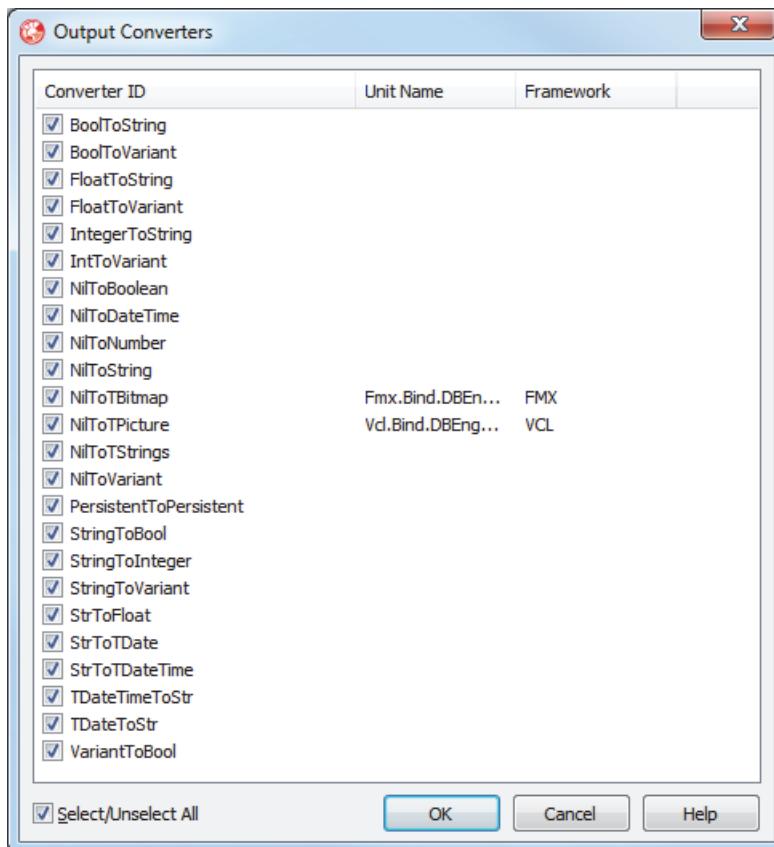
V ogrodje LiveBindings lahko dodajamo tudi lastne funkcije, a je to le malokrat potrebno. Izraz lahko namreč uporablja tudi funkcije objekta, s katerim je povezan (ter preko ovinka z lastnostjo Owner tudi funkcije obrazca). Če bi radi dodali funkcijo, jo samo implementiramo v obrazcu in uporabimo v izrazu. V našem primeru je takšna funkcija NowStr.

```
function TfrmXE2SimpleBindings.NowStr: string;
begin
  Result := FormatDateTime('ttt', Now);
end;
```

Opozorili bi radi še na dejstvo, da *izraz* ni ekvivalent *žive povezave*. V zgornjem primeru je sicer vedno veljalo, da je ena živa povezava vsebovala en izraz, v splošnem pa ni nujno tako. Drugi tipi živih povezav, ki si jih bomo ogledali v nadaljevanju, lahko vsebujejo več izrazov, ki jih izvedejo zaporedno.

Izhodni pretvorniki

Izhodni pretvorniki (output converters) so v ogrodje LiveBindings vgrajeni pretvorniki, ki samodejno pretvarjajo vrednosti, kadar te ne ustrezajo ciljni lastnosti. Zaradi tega lahko na primer neposredno in brez uporabe funkcij povežemo izvorno lastnost TSpinEdit.Value, ki je tipa Integer, z lastnostjo TEdit.Text, ki je tipa String. Za pretvorbo bo poskrbel pretvornik IntegerToString. Tudi nabor pretvornikov lahko razširimo z lastnimi funkcijami.



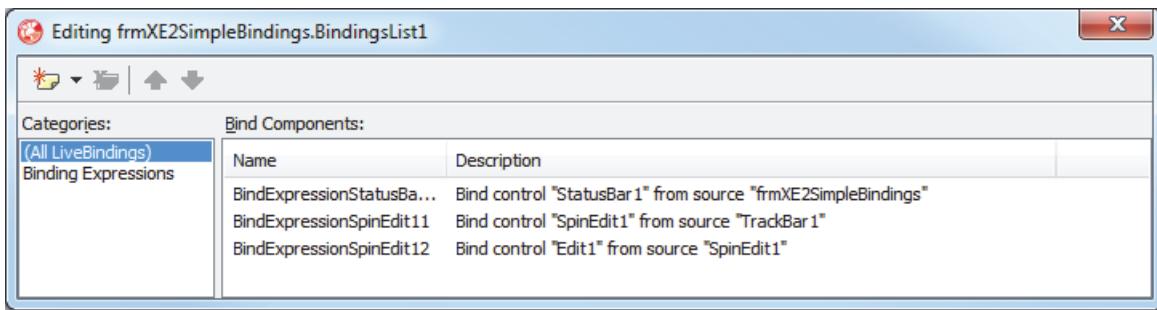
LiveBindings komponente

Oglejmo si sedaj komponente, ki so del ogrodja LiveBindings in jih bomo pri delu z njim nenehno srečevali.

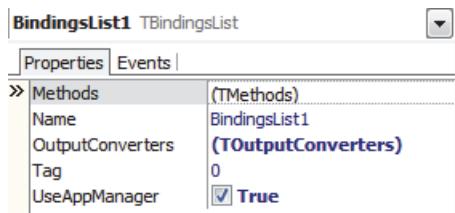
TBindingsList

TBindingsList je vsebnik (container) za žive povezave. Vsaka živa povezava je shranjena v vsebniku tega tipa. Ko na obrazec postavite prvo živo povezavo, RAD Studio samodejno naredi komponento tipa TBBindingsList in jo postavi na obrazec, vanjo pa shrani tudi vse kasneje uporabljene žive povezave.

Ob dvojnem kliku komponente se pojavi urejevalnik, ki prikaže seznam živih povezav. Poleg dostopa do njih ob enojnem ali dvojnem kliku (kar smo opisali v uvodnem delu) omogoča urejevalnik še dodajanje, brisanje in razvrščanje živih povezav. Vrstni red je v redkih primerih pomemben, ker vpliva na vrstni red obdelave (izračunavanja in prirejanja) živih povezav.



TBindingsList združuje tudi vse metode in izhodne pretvornike. S klikom gumba s tropičjem v lastnostih Methods in OutputConverters si lahko ogledamo nameščene metode in pretvornike in jih po želji tudi onemogočimo.



Pomemben del komponente je metoda Notify, ki smo jo v zgornjem primeru že spoznali in sproži osveževanje živih povezav, povezanih z določenim objektom.

Na obrazcu imate lahko več hkratnih komponent TBbindingsList, vendar bo to le redko potrebno.

TBindScope in TBindScopeDB

Komponenti TBindScope in TBindScopeDB »ovijeta« nek drug razred (TBindScope poljuben objekt, TBindScopeDB pa podatkovno bazo, oziroma bolj natančno TDataSource) in omogočita izrazu žive povezave, da dostopa do delov objekta, do katerih sicer ne bi imel dostopa. TBindScopeDB je še posebej pomemben, ker olajša dostop do polj podatkovne baze.

Povezovalni razredi

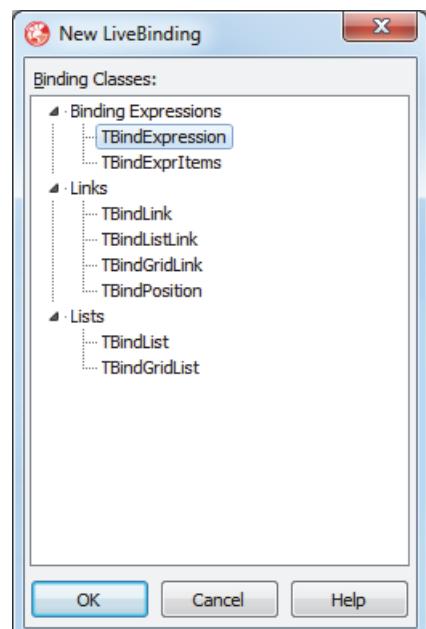
Ko naredimo novo živo povezavo, RAD Studio vpraša, kakšen naj bo njen tip. Ponudi nam osem različnih razredov – dva v skupini *Binding Expressions*, štiri v skupini *Links* in dva v skupini *Lists*.

TBindExpressions

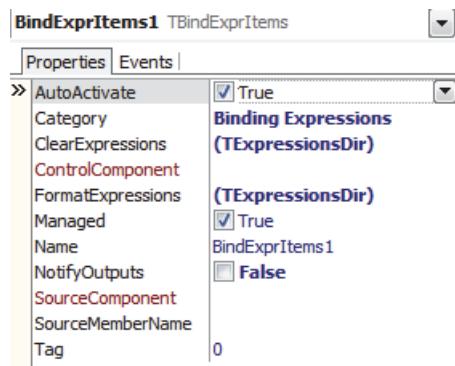
Razred TBindExpressions smo že dodobra spoznali v uvodnem primeru. Namenjen je povezovanju dveh lastnosti dveh gradnikov (z enosmernim ali dvosmernim prenosom podatkov) ali povezovanju lastnosti gradnika z izrazom, ki izračuna neko vrednost. Oba načina rabe smo že spoznali in sta predstavljena v programu XE2SimpleBindings.

TBindExprItems

Razred TBindExprItems je podoben razredu TBindExpressions, vendar omogoča, da na eno živo povezavo povežete več različnih izrazov. V ta namen sta lastnosti SourceExpression in ControlExpression zamenjani z lastnostjo FormatExpressions, v kateri lahko navedemo več izrazov,

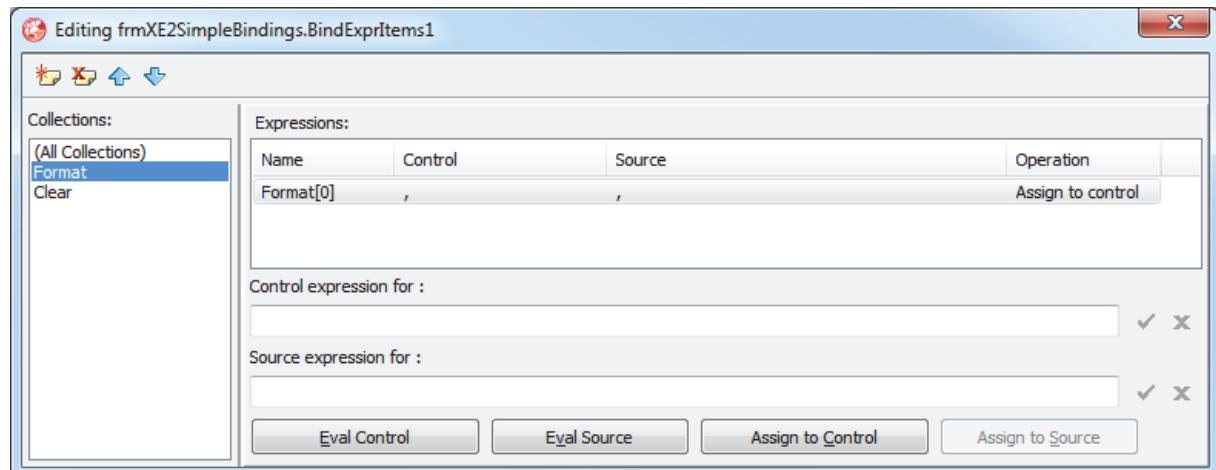


vsak pa ima svoj vir (SourceExpression), cilj (ControlExpression) in smer (Direction).



Poleg tega imamo na voljo še seznam ClearExpressions, v katerem prav tako lahko navedemo več izrazov. Ti izrazi se izvedejo, kadar vir podatkov ni več na voljo. Tipičen primer za to je povezava s podatkovno bazo. Če vir podatkov deaktiviramo (na primer z ClientDataSet.Active := false), se bodo izvedli izrazi v seznamu ControlExpressions in (na primer) počistili vsebino ciljnega objekta.

Še najlažje urejamo kontrolne izraze, tako da v komponenti TBindingsList poiščemo živi izraz in ga dvokliknemo. Odprl se bo urejevalnik, v katerem lahko dodajate in brišete obe vrsti izrazov – Format in Clear.



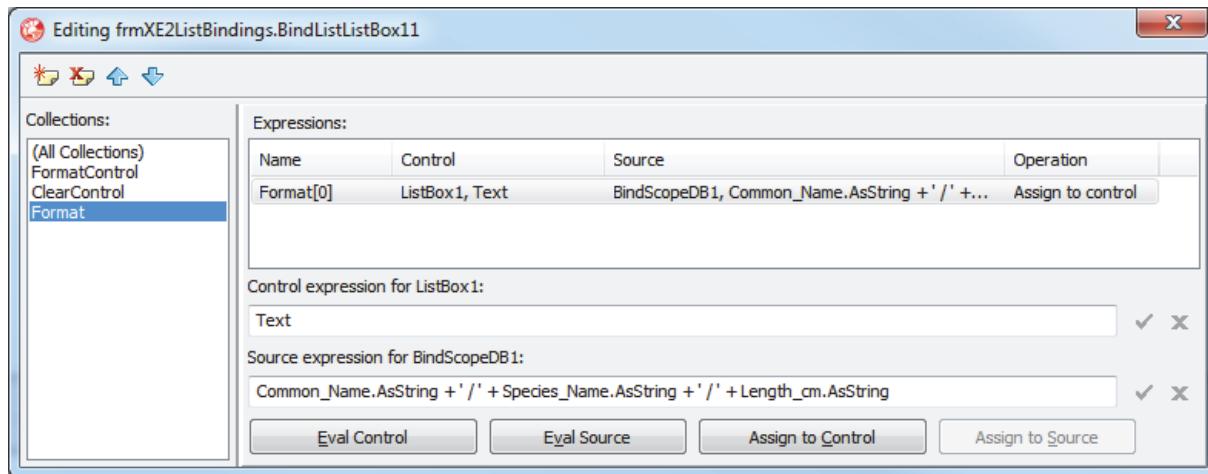
TBindList

Oba razreda v razdelku Lists sta namenjena povezavam, kjer lahko vir posreduje seznam podatkov. Tipičen primer za tako obnašanje je podatkovna baza. Razred TBindList je namenjen povezovanju na sezname (TListBox, TComboBox, TListView in TStringGrid), razred TBindGridList pa povezovanju na preglednice (TStringGrid). Oba razreda sta uporabljeni v primeru XE3ListBindings.

Razred TBindList vedno predstavlja enosmerno povezavo – podatki tečejo le v smeri od SourceComponent do ControlComponent. Izraze lahko zapišemo v tri sezname.

1. FormatControl – izrazi, ki se izvedejo ob aktivni izvorni komponenti in vplivajo na ciljni gradnik.
2. Format – izrazi, ki se izvedejo ob aktivni izvorni komponenti in vplivajo na *seznam* v ciljnem gradniku. Ciljna lastnost (property) je odvisna od vrste ciljnega gradnika; v primeru gradnika TListBox bodo ti izrazi vplivali na lastnost Items.

3. ClearControl – izrazi, ki se izvedejo ob neaktivni izvorni komponenti.



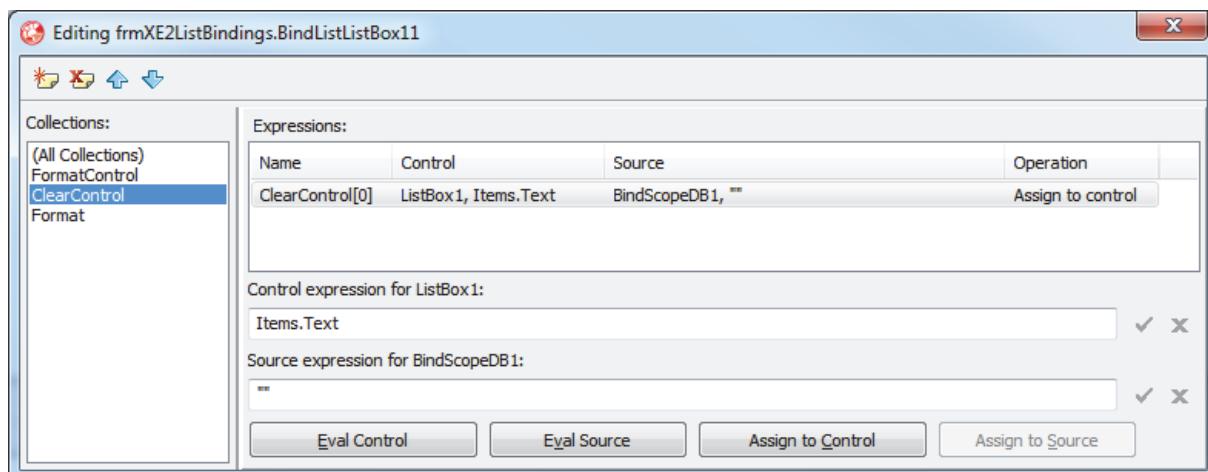
Žive povezave tega tipa bomo skoraj vedno uporabljali z virom tipa TBindScopeDB. Takšen primer je uporabljen tudi v programu XE2ListBindings.

Na obrazcu imamo komponento ClientDataSet1, ki vsebuje podatke iz baze *Fish Facts*, komponento DataSource1, ki ima za DataSet nastavljen ClientDataSet1 in komponento BindScopeDB1, ki ima za DataSource nastavljen DataSource1.

Na obrazcu je tudi gradnik ListBox1, povezan z živo povezavo BindListListBox11, ki ima definiran en izraz v seznamu Format (zgornja slika) ter en izraz v seznamu ClearControl (spodnja slika).

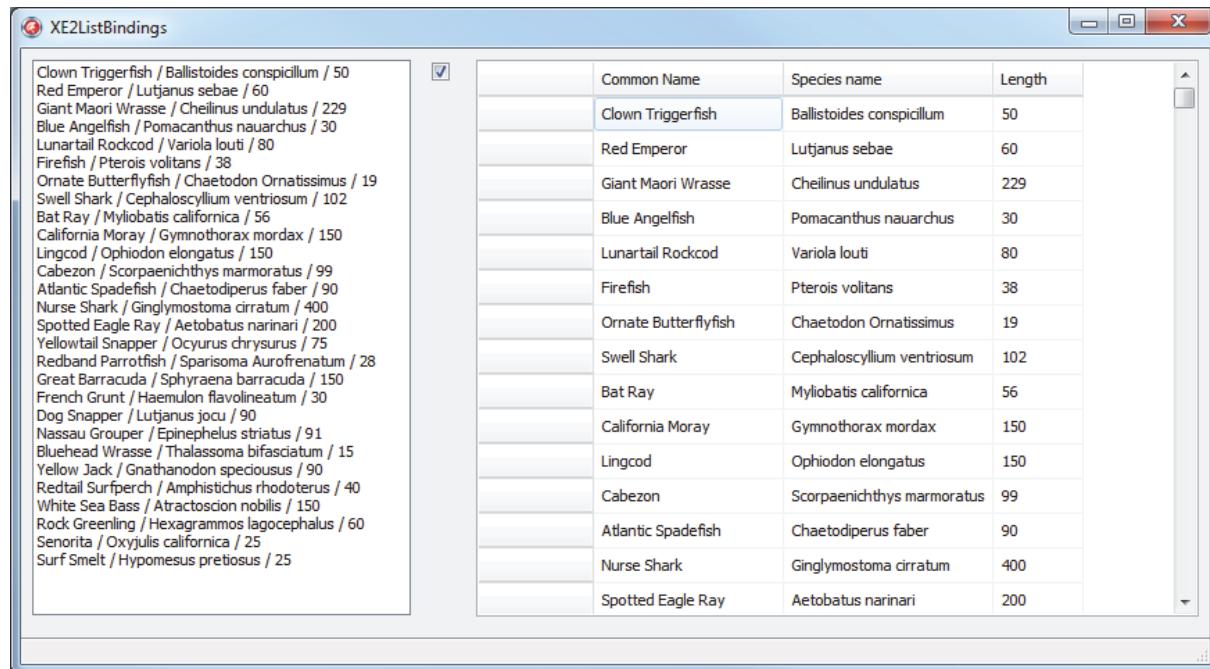
Izraz v seznamu Format vzame tri polja iz baze ter jih združi v niz. Komponenta BindScopeDB1 poskrbi, da lahko do polj iz baze dostopamo kar z njihovim imenom (v našem primeru Common_Name, Species_Name in Length_cm). Živa povezava tipa TBindList pa poskrbi, da se izraz izvede po enkrat za vsak element vhodnega seznama (baze) in zapiše v seznam izhodnega gradnika.

Izraz v seznamu ClearControl je enostavnejši, le Items.Text postavi na prazno vrednost.



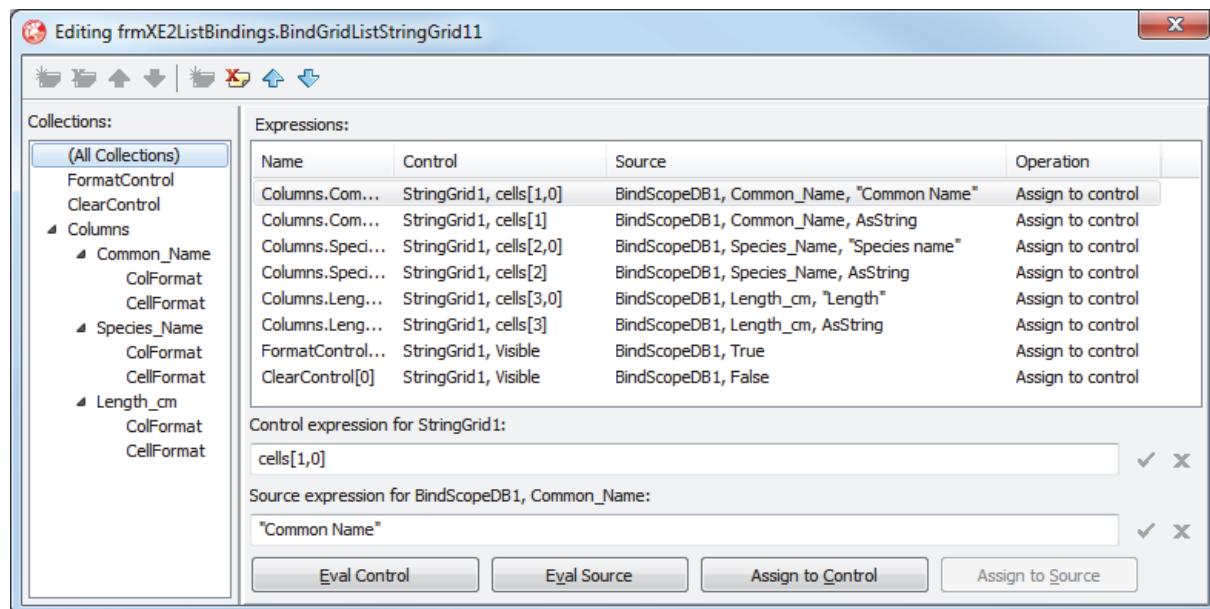
Opazili boste, da smo v prvem primeru (Format) za ciljni izraz nastavili Text, v drugem pa Items.Text. Do razlike pride zaradi tega, ker izrazi v seznamu Format vplivajo na TListBox.Items, izrazi v seznamu ClearControl pa na TListBox.

Če program poženemo, vidimo, da se v seznamu ListBox1 pojavijo vsa polja iz podatkovne baze.

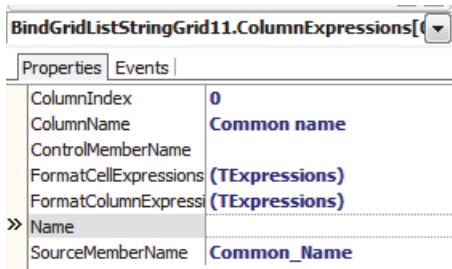


TBindGridList

Povezave tipa TBindGridList so namenjene dvosmernemu povezovanju seznamov podatkov (običajno bo to podatkovna baza) s preglednico (TStringGrid). Delo z njimi pa je v okolju VCL strašno okorno; uporabni postanejo šele v okolju FireMonkey.



Poleg seznamov FormatControl in ClearControl imamo v razredu TBindGridList še seznam Columns, v katerega dodajamo objekte, ki opisujejo posamezne stolpce v preglednici.



Vsek stolpec z lastnostjo SourceMemberName povežemo s poljem v bazi, nato pa mu nastavimo izraze FormatColumn (izvedejo se enkrat za vsak stolpec in nastavijo lastnosti stolpca) ter izraze FormatCell (izvedejo se za vsako vrstico v preglednici).

TBindLink

Razredi v skupini Links so podobni nekaterim zgoraj opisanim razredom, le da so močno izboljšani. Vsi so dvosmerni in podpirajo več hkratnih izrazov. Poleg tega so vsi vrste *unmanaged*, torej delujejo pravilno le z gradniki, ki implementirajo vmesnika IEditLinkObserver ali IEditGridLinkObserver.

TBindLink je različica razreda TBindExprItems. Vsebuje sezname izrazov Format (generiranje izhodne vrednosti pri aktivni komponenti), Clear (generiranje izhodne vrednosti pri neaktivni komponenti) ter Parse (branje izhodne vrednosti z namenom generiranja vhodne vrednosti). Slednji izrazi omogočajo dvosmerno povezavo. Iz ciljnega gradnika znajo izločiti podatke, nato pa LiveBindings poskrbi, da se s temi podatki osveži vhodni (Source) gradnik.

TBindListLink

TBindListLink je različica razreda TBindList. Vsebuje sezname ClearControl, FormatControl, Format (ti delujejo tako kot v TBindList), Parse (deluje tako kot v TBindLink) ter PosControl in PosSource (delujeta tako kot v TBindPosition, ki ga bomo še opisali).

TBindGridLink

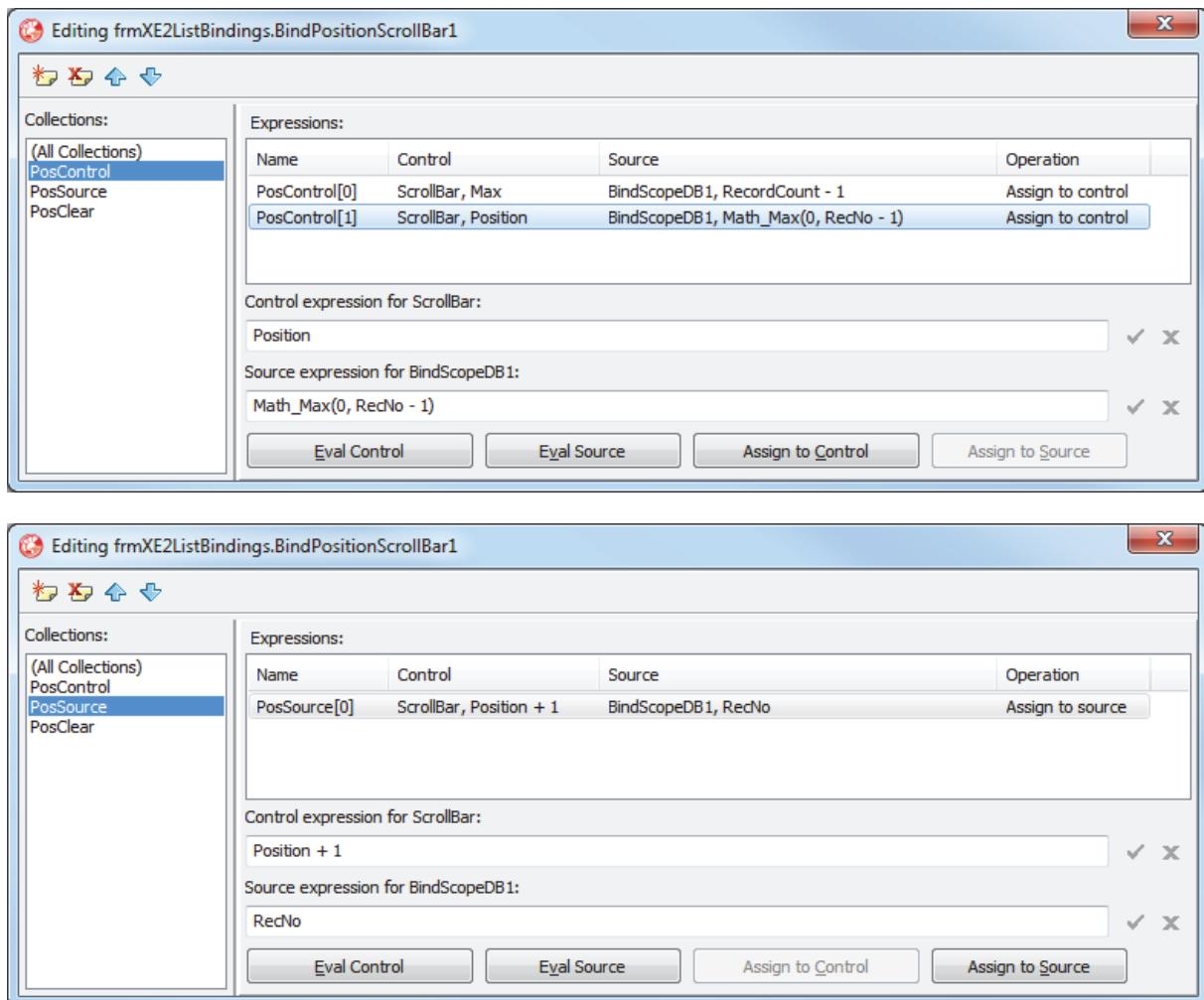
TBindListLink je različica razreda TBindGridList. Vsebuje sezname ClearControl, Column, FormatControl, PosControl in PosSource. Elementi seznama Column vsebujejo sezname FormatCell, FormatColumn in ParseCell.

TBindPosition

Razred je namenjen povezovanju z gradniki, ki implementirajo vmesnik IPositionObserver. Tipičen primer je gradnik TScrollBar.

Razred vsebuje tri sezname izrazov. PosClear se izvedejo, kadar se izvorna komponenta deaktivira. PosSource se izvedejo, kadar se spremeni lastnost Position (ozioroma ekvivalentna lastnost z drugim imenom) izvirne komponente, PosControl pa, kadar se spremeni lastnost Position ciljne komponente.

Testni program prikazuje, kako lahko položaj v podatkovni bazi (BindScopeDB.RecNo) povežemo s položajem drsnika (TScrollControl.Position)



Povezovanje z objekti

Omenili smo že, da lahko LiveBindings uporabimo tudi za povezovanje uporabniškega vmesnika z objekti, ki jih zgradimo v programu in nimajo vizualne reprezentacije. Tak način rabe je še posebej primeren za rabo v programih, ki imajo vmesnik strogo ločen od poslovne logike.

Povezovanje z objekti je prikazano v programu XE2BindToObject.

Oglejmo si primer, kjer imamo definiran enostaven poslovni objekt TPerson, ki vsebuje le polje Name.

```
type
  TPerson = class(TBoundObject)
  private
    FName: string;
    procedure SetName(const AValue: string);
  public
    property Name: string read FName write SetName;
  end;
```

Objekt je izpeljan iz razreda TBoundObject, ki ga je napisal Jarrod Hollingworth in predstavil na strani <http://members.adug.org.au/2012/03/16/using-livebindings-to-connect-the-ui-to-objects/>. Bistvo razreda TBoundObject je, da v metodi Bind naredi dvosmerno povezavo z lastnostjo drugega objekta, ki jo specificira programer.

```
procedure TBoundObject.Bind(const AProperty: string;
  const ABindToObject: TObject; const ABindToProperty: string;
  const ACreateOptions: TBindings.TCreateOptions);
begin
  // From source to dest
  FBindings.Add(TBindings.CreateManagedBinding(
    { inputs }
    [TBinds.CreateAssociationScope([Associate(Self, 'src')])],
    'src.' + AProperty,
    { outputs }
    [TBinds.CreateAssociationScope([Associate(ABindToObject, 'dst')])],
    'dst.' + ABindToProperty,
    nil, nil, ACreateOptions));
  // From dest to source
  FBindings.Add(TBindings.CreateManagedBinding(
    { inputs }
    [TBinds.CreateAssociationScope([Associate(ABindToObject, 'src')])],
    'src.' + ABindToProperty,
    { outputs }
    [TBinds.CreateAssociationScope([Associate(Self, 'dst')])],
    'dst.' + AProperty,
    nil, nil, ACreateOptions));
end;
```

V naši kodi moramo objekt le narediti in njegovo lastnost Name »pripeti« na inpName.Text.

```
procedure TfrmBindToObject.FormCreate(Sender: TObject);
begin
  FPerson := TPerson.Create;
  FPerson.Name := 'Janez';
  FPerson.Bind('Name', inpName, 'Text');
end;
```

Testni program ob spremembi polja inpName.Text in ob spremembi lastnosti FPerson.Name obvesti ogrodje LiveBindings o spremembi, tako da lahko ogrodje izračuna potrebne izraze in osveži povezane lastnosti.

```
procedure TPerson.SetName(const AValue: string);
begin
  if AValue <> FName then begin
    FName := AValue;
    Notify('Name');
  end;
end;

procedure TfrmBindToObject.inpNameChange(Sender: TObject);
begin
  TBindings.Notify(Sender, 'Text');
end;
```

LiveBindings in FireMonkey

Ogrodje FireMonkey ne vsebuje podatkovno osveščenih (data-aware) gradnikov, zato je LiveBindings najlažji način povezave baze podatkov z vizualnimi gradniki. Ugotovili smo že, da je povezovanje s preglednicami (grid) v okolju VCL izredno nerodno, zato si lahko kar oddahnemo, ko ugotovimo, da ima FireMonkey povezovanje z bazami podatkov bistveno boljše rešeno.

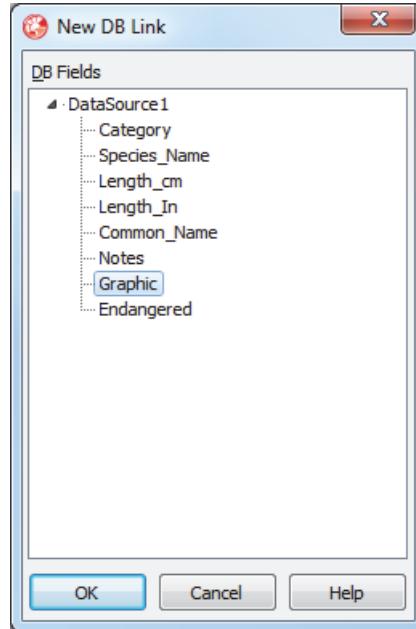
FireMonkey prinaša kopico dodatnih tipov živih povezav: TBindDBEditLink, TBindDBTextLink, TBindDBCheckLink, TBindDBGridLink, TBindDBListLink, TBindDBImageLink in TBindDBMemoLink. Namenjene so povezovanju s polji v podatkovnih bazah.

Dodana je tudi komponenta BindNavigator, ki je namenjen navigaciji po bazi podatkov in zna sodelovati z ogrodjem LiveBindings.

V ogrodju FireMonkey gradnik povežemo z bazo podatkov, tako da iz pojavnega menuja ali iz menuja desno od lastnosti LiveBindings izberemo Link to DB Field...

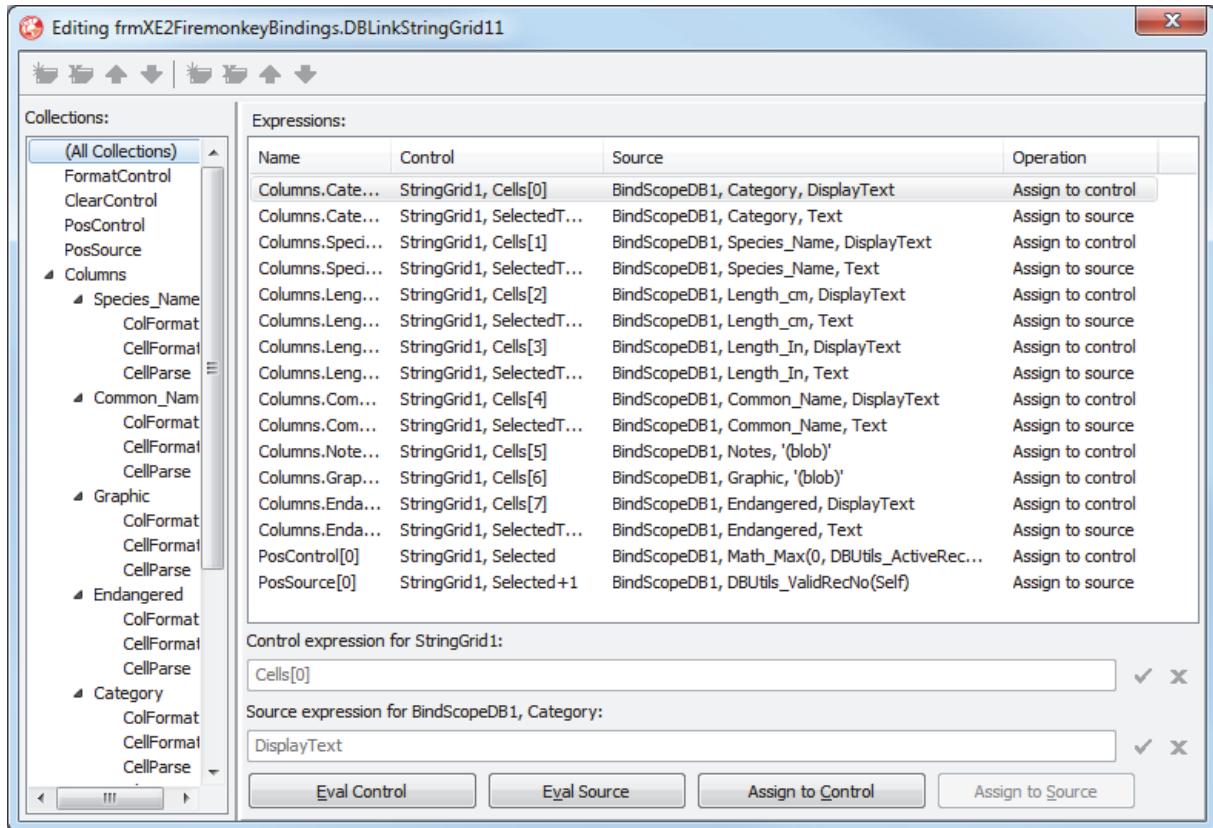


RAD Studio samodejno naredi komponento tipa TBindScopeDB in jo poveže s komponento TDataSource na obrazcu. Nato prikaže seznam polj iz baze. Izberemo želeno polje, kliknemo OK in delo je končano.



RAD Studio na obrazec samodejno postavi komponenti TBindingsList in TBindScopeDB ter slednjega poveže z virom podatkov (TDataSource).

Enostavno je tudi povezovanje preglednic. Na obrazec postavimo TStringGrid, kliknemo z desno tipko in izberemo Link to DB DataSource. RAD Studio pokaže vse primerne vire podatkov (TBindScopeDB). Izberemo pravega in potrdimo izbiro. RAD Studio bo izdelal povezavo tipa TBindDBGridLink in napolnil vse njene lastnosti.



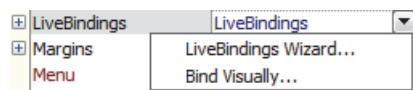
Nerodno je le, da tako izdelanih povezav ne moremo ročno spremenjati. Če odpremo urejevalnik (zgornja slika), vidimo, da so vse možnosti urejanja ugasnjene. Stanje je (žal) enako tudi v RAD Studio XE3 – če povezavo naredimo s čarovnikom ali vizualnim povezovanjem, njenih lastnosti ne moremo spremenjati v urejevalniku.

Visual LiveBindings v RAD Studiu XE3

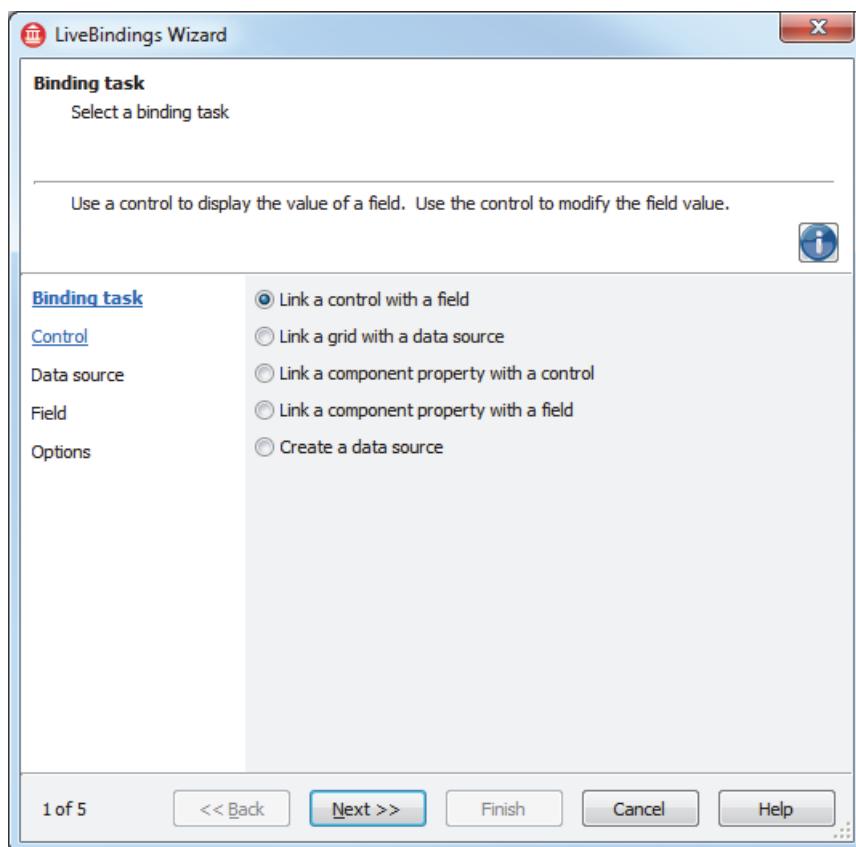
RAD Studio XE3 je ogrodje LiveBindings izboljšal na več področjih. Dodali so vizualno povezovanje gradnikov (Visual LiveBindings), novega čarovnika za povezovanje, nekaj novih komponent (TAdapterBindSource, TPrototypeBindSource, TBindSourceDB, TBindSourceDBX), gradnik TBindNavigator pa je po novem dostopen tudi v ogrodju VCL.

Čarovnik za povezovanje

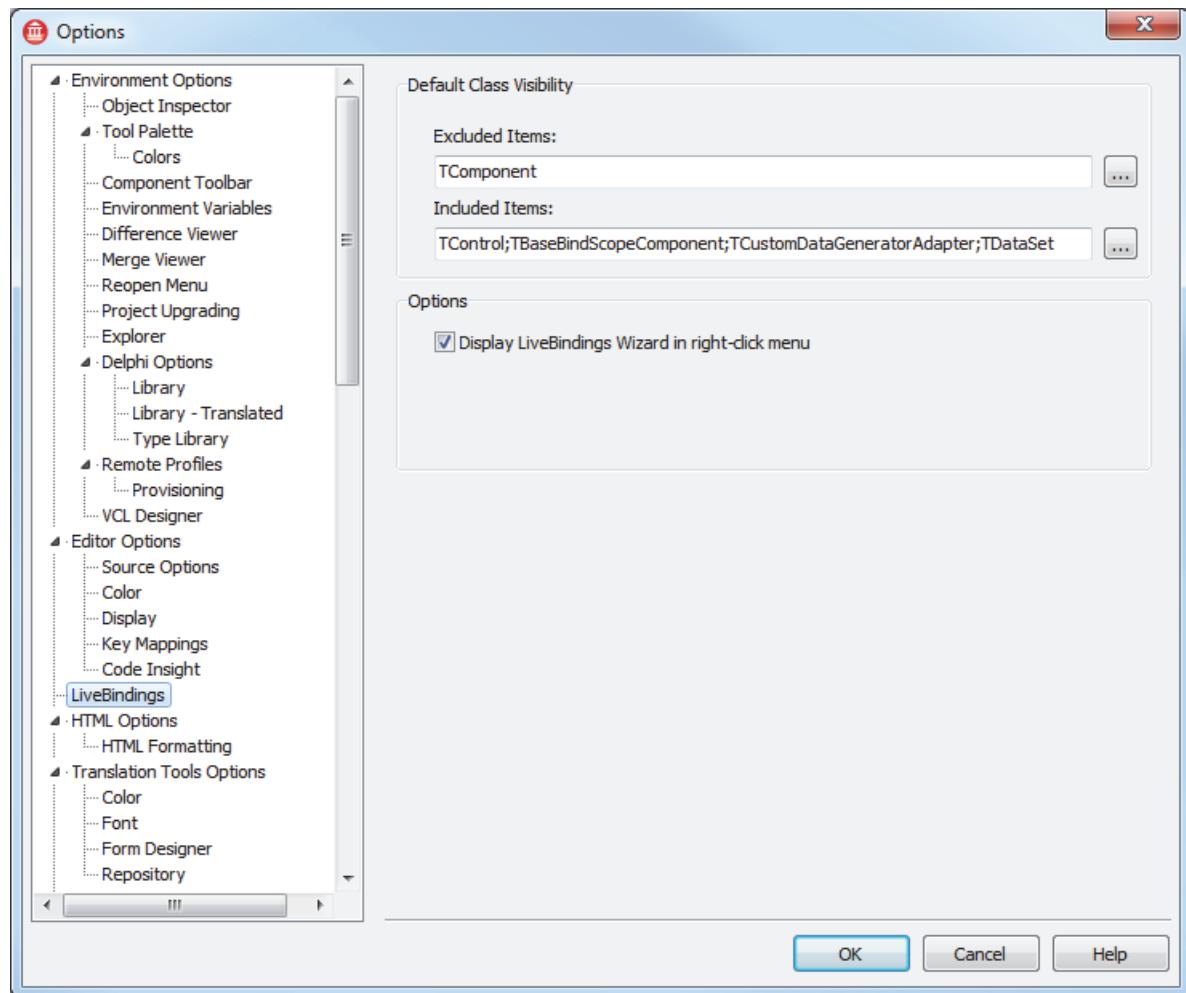
V RAD Studiu XE2 ustvarimo novo živo povezavo z izbiro New LiveBinding v pojavnem menuju ali v menuju lastnosti LiveBindings. V XE3 imamo na volj dve možnosti – LiveBindings Wizard in Bind Visually.



LiveBindings Wizard je zamenjal izbiro New LiveBinding. Ob kliku dobimo na zaslon čarovnika, ki nas sprehodi skozi različne možnosti povezav. Izberi možnosti na prvem zaslonu je prirejena trenutno izbranemu gradniku, zato v večini primerov ne boste videli vseh petih možnosti iz spodnje slike.

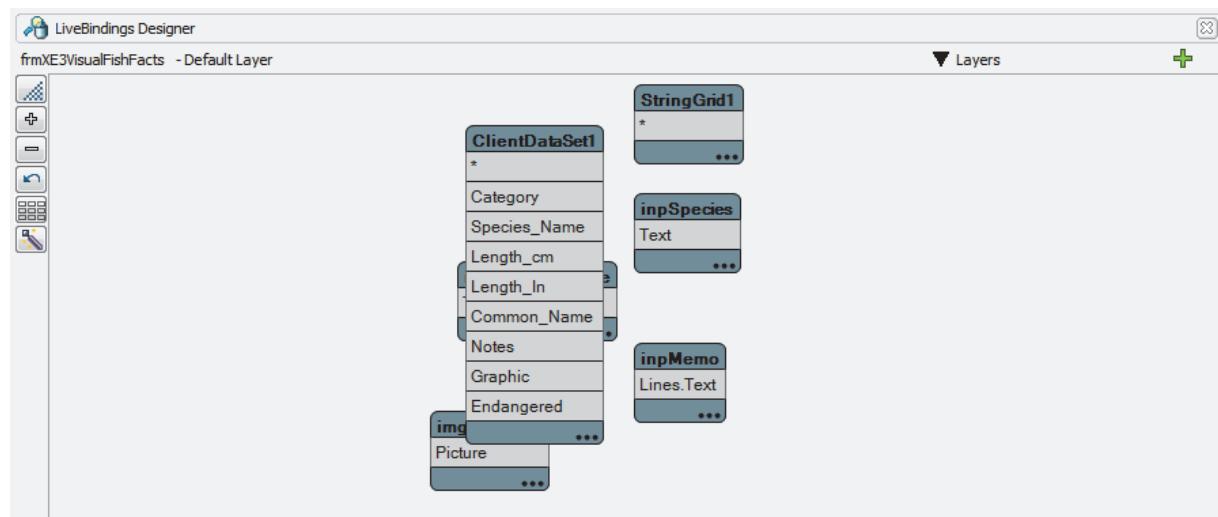


Da bi bil čarovnik dostopen tudi skozi pojavní menu (ta se prikaže, ko z desno mišjo tipko kliknete obrazec ali gradnik), morate v nastavitevah prižgati možnost »Display LiveBindings Wizard in right-click menu« v veji LiveBindings.



Vizualno povezovanje

Druga možnost – Bind Visually – odpre grafični urejevalnik LiveBindings Designer. V njem se pojavijo vsi objekti, ki vsebujejo vsaj eno povezljivo lastnost, ter žive povezave med njimi. Urejevalnik je privzeto priklopljen na dno zaslona, lahko pa ga tudi »odpnemo« in premaknemo kadarkoli. Če uporabljate več kot en monitor, priporočamo, da LiveBindings Designer odklopite, povečate in premaknete na sekundarni monitor.

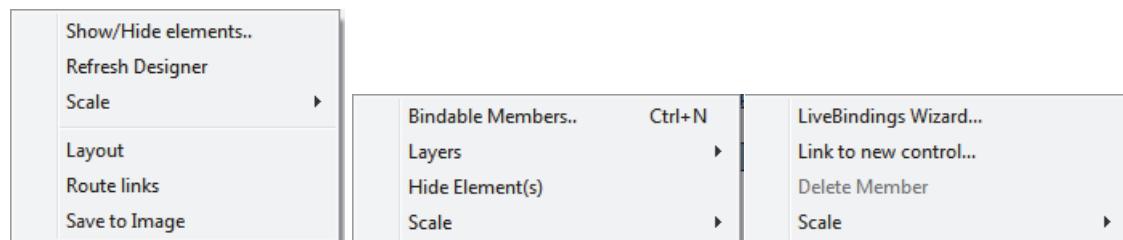


Ob prvem priklicu znajo biti objekti precej nametani na zaslonu. Lahko jih uredite ročno (kliknete na zaobljeni pravokotnik, ki predstavlja vizualni gradnik, in ga potegnete kamorkoli), ali pa uporabite ikono za samodejno urejanje (petta ikona od zgoraj v stolpcu ikon na levi).

Ikone imajo sledeči pomen (od zgoraj navzdol):

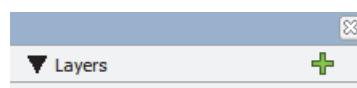
- Povečaj/pomanjšaš sliko, da bodo vsi objekti vidni v oknu urejevalnika.
- Povečaj (približaj) sliko.
- Pomanjšaj (oddalji) sliko.
- Nastavi privzeto velikost prikaza.
- Razporedi objekte.
- Prikliči čarownika za povezovanje.

Veliko ukazov je dostopnih tudi skozi pojavnne menuje, ki se bistveno razlikujejo glede na mesto klica (prazni del urejevalnika, ime objekta, lastnost objekta). Nekaj primerov si lahko ogledate na spodnji sliki.



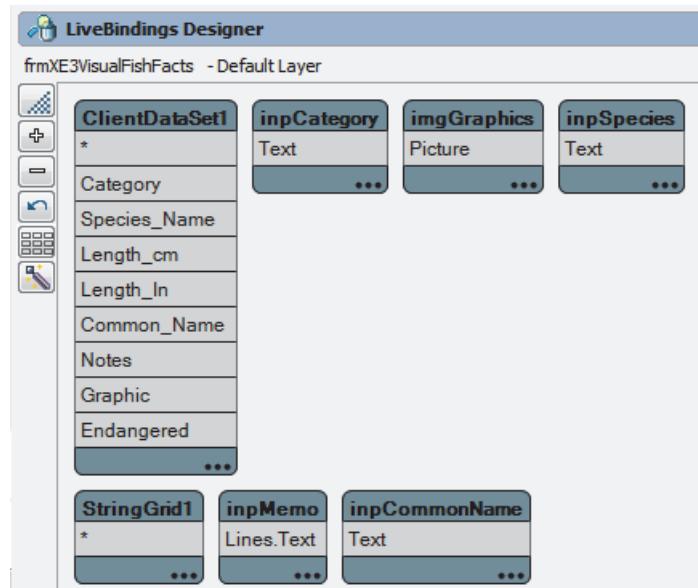
Zelo uporabna izbira je Bindable Members, s katero lahko določimo lastnosti objekta, ki bodo vidne v vizualne urejevalniku. Privzeto so namreč vidne le lastnosti, ki jih najpogosteje uporabimo za povezovanje.

Na bolj zapletenih obrazcih lahko gradnike razvrstimo v sloje (layers). Ukazi za delo s sloji se skrivajo v zgornjem desnem kotu vizualnega urejevalnika.



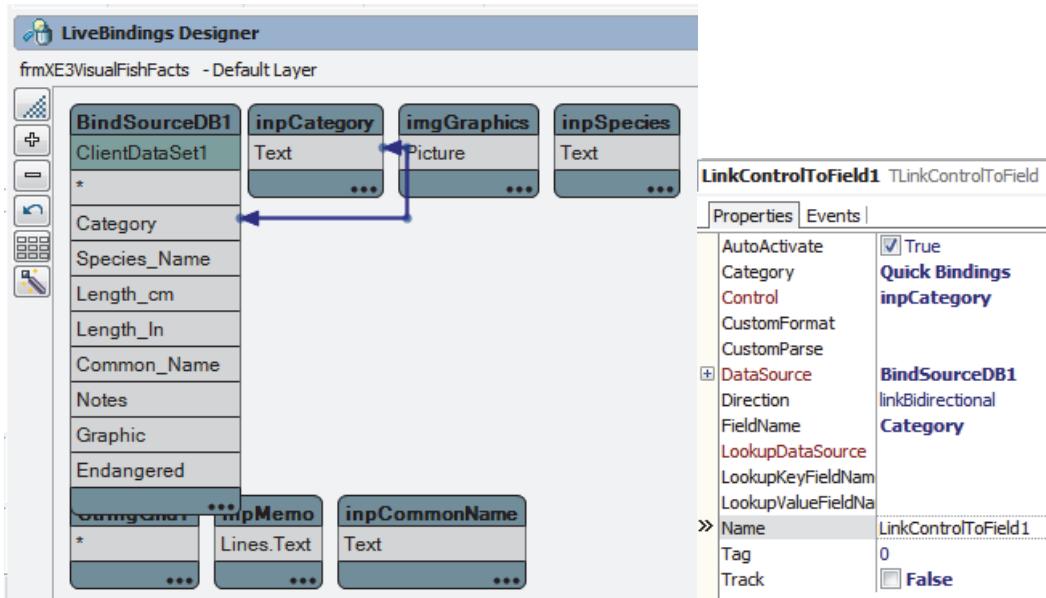
Delovanje urejevalnika si bomo ogledali na primeru. Odprimo testni program XE3VisualFishFacts. Znašli se bomo pred obrazcem z nekaj gradniki brez vsebine ter podatkovno komponento ClientDataSet, v katero so že naloženi podatki iz baze *Fish Facts*.

Odprimo vizualni urejevalnik in z ikono razporedimo objekte.



Povezovanje začnimo v polju **inpCategory** (vnosno polje TEdit, ki je na obrazcu postavljeno levo zgoraj). Kliknimo v polje Text in (ne da bi izpustili levo tipko miške) potegnimo povezavo v polje **Category** objekta ClientDataSet. Nato izpustimo levo tipko. Vizualni urejevalnik potegne povezavo med izbranimi lastnostma.

Povezava v vizualnem urejevalniku predstavlja živo povezavo. Če jo izberemo (kliknemo), lahko v Object Inspectorju nastavimo njene lastnosti. Povezavo lahko izbrišemo z desnim klikom in izbiro možnosti Remove Link.

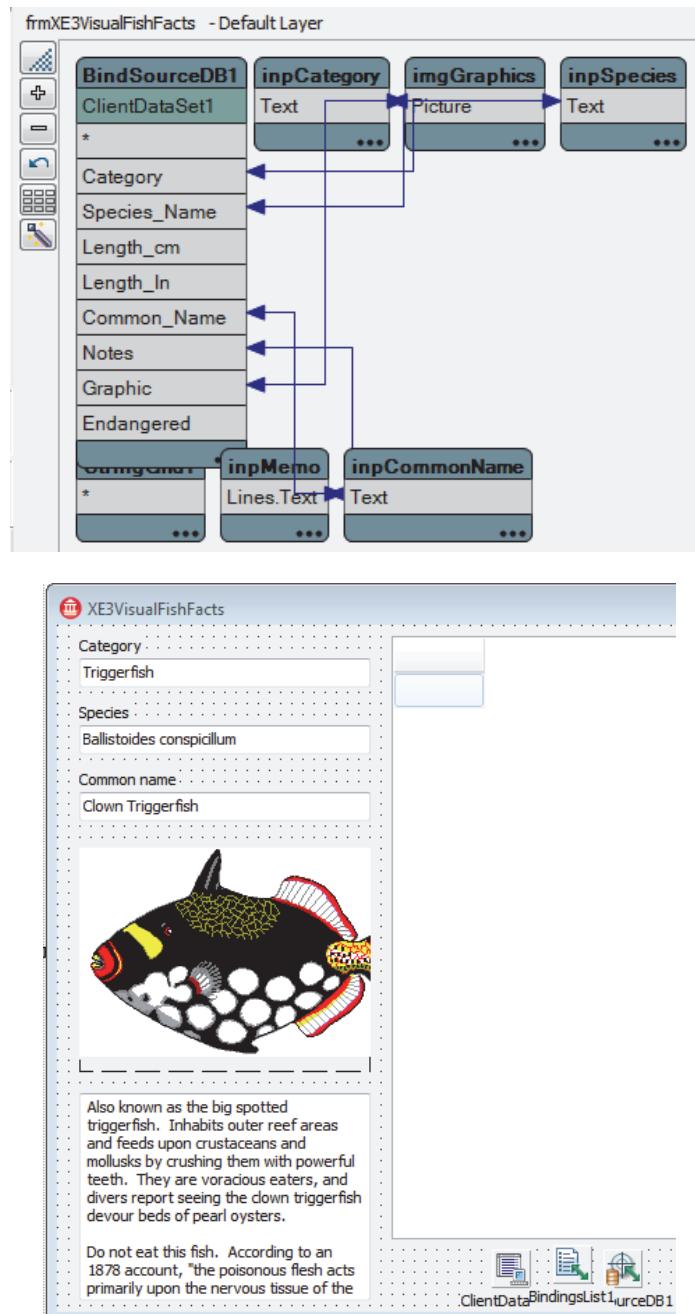


Takoj ko ustvarimo živo povezavo, se na zaslonu v polju **inpCategory** že pojavijo podatki iz prvega polja v bazi.

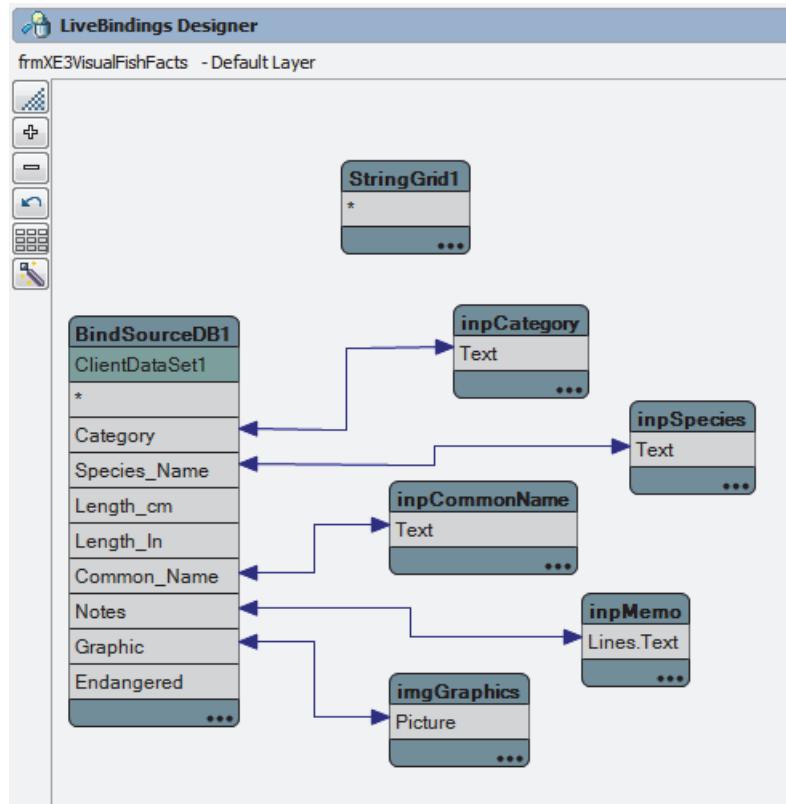


Hkrati s prvo povezavo je RAD Studio ustvaril še gradnika TBindingsList in TBindSourceDB. Slednji je zamenjava za TBindScopeDB, ki se zna priključiti neposredno na *dataset* (torej ne potrebujemo posrednika v obliki komponente TDataSource).

Povežimo sedaj še ostanek polj – inpSpecies.Text s ClientDataSet1.Species_Name, inpCommonName.Text s ClientDataSet1.Common_Name, imgGraphics.Picture s ClientDataSet1.Graphics in inpMemo.Lines.Text s ClientDataSet1.Notes.

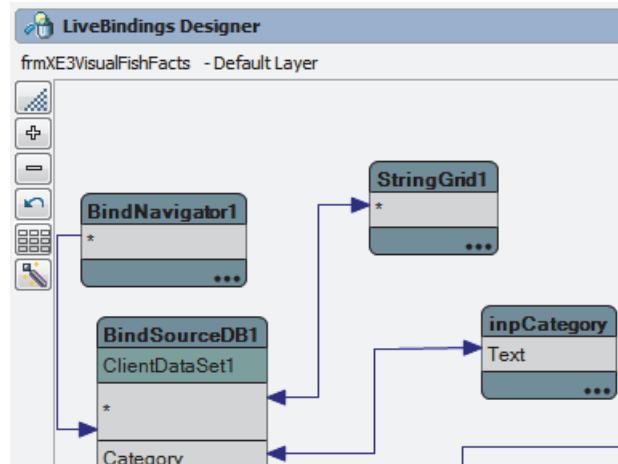


V vizualnem urejevalniku vlada že precejšnja zmeda, zato objekte lepše razporedimo z miško. Samodejno razporejanje (s klikom ikone ali izbiro Layout v pojavnem menuju) žal ne naredi uporabnega rezultata.

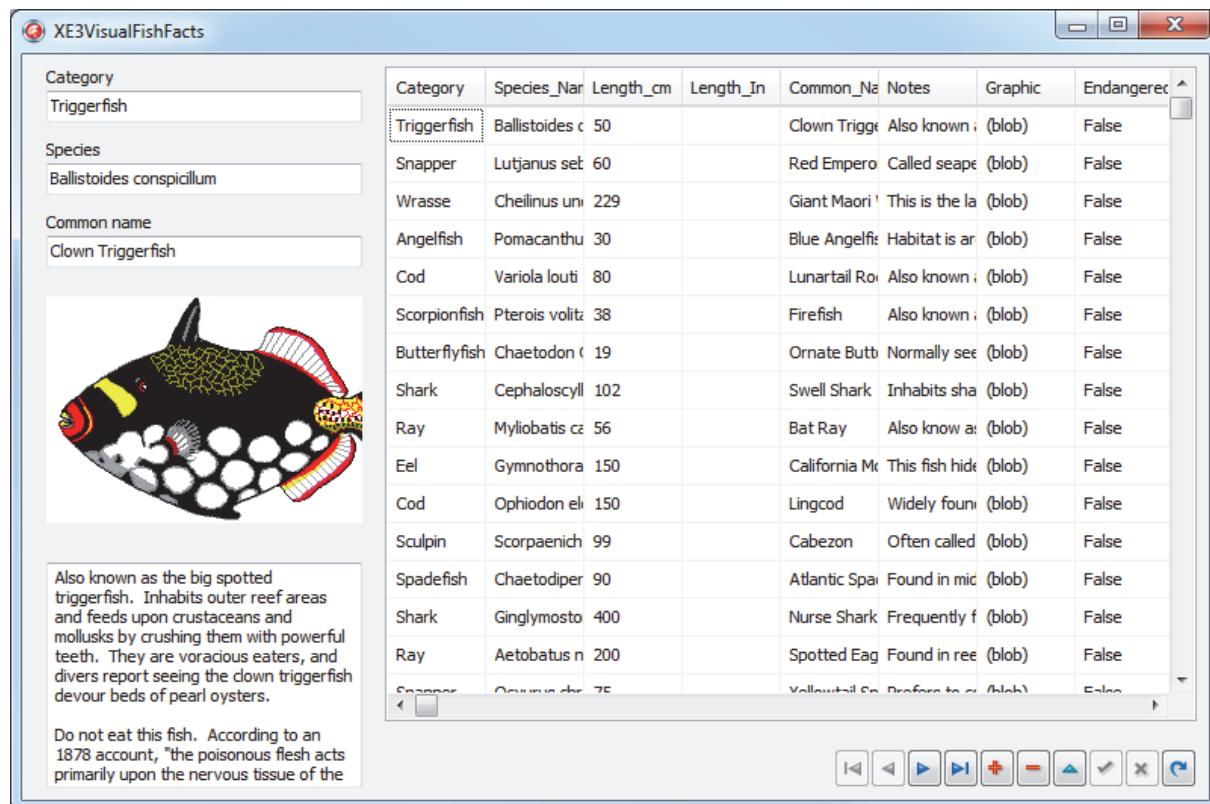


Povezati moramo še preglednico TStringGrid. Ker bi radi v njej prikazali vsa polja iz baze, potegnemo črto iz StringGrid1.* na ClientDataSet1.*.

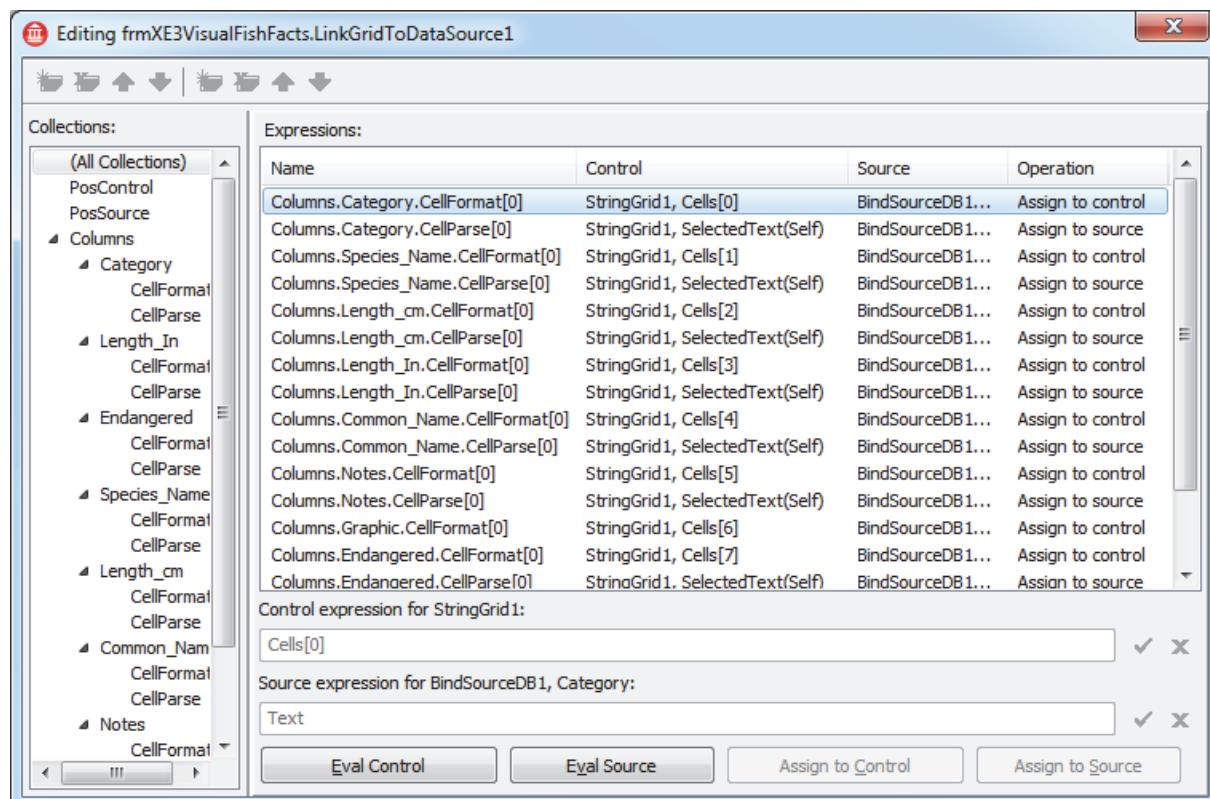
Dodajmo še navigacijsko komponento. Na obrazec odložimo komponento TBindNavigator ter v vizualnem urejevalniku BindNavigator1.* povežemo z StringGrid1.*.



Program je končan. Samo še poženemo ga.



Žal samodejno ustvarjenih povezav in izrazov – tako kot v RAD Studiu XE2 – ne moremo urejati. Lahko jih samo izbrišemo.



CustomFormat

Žive povezave so v RAD Studiu XE3 doobile novo lastnost CustomFormat. V njej lahko preoblikujemo podatek, ki je prispel po živi povezavi, preden se prikaže v gradniku. Pri tem lahko uporabimo enake funkcije, kot so na voljo za pisanje izrazov.

Oglejmo si, kako bi ime kategorije ribe v našem primeru spremenili v velike črke.

Izberemo povezavo do ClientDataSet1.Category. Nato v Object Inspectorju v polje CustomFormat vpišemoUpperCase(%s). Simbol %s v tem primeru predstavlja vrednost, ki je prispela po živi povezavi.



Na voljo nam je tudi lastnost CustomParse, v katero lahko vpišemo izraz, ki bo prebral vrednost podatka iz ciljnega gradnika, preden ga po dvosmerni povezavi pošljemo v nasprotno smer.

TAdapterBindSource

Komponenta TAdapterBindSource olajša povezovanje z objekti, ki jih ustvarimo v programu. To smo lahko počeli že v RAD Studiu XE2 (poglavlje *Povezovanje z objekti*), le da je bilo treba natipkati precej kode. V RAD Studiu XE3 je delo lažje, povezujemo pa se lahko tudi vizualno.

Prikažimo rabo te komponente na testnem primeru (XE3AdapterBindSource; primer je prepisani s spletni strani <http://www.malcolmgroves.com/blog/?p=1084>). V programu imamo definiran razred TPerson, ki opisuje osebo.

```
type
  TPerson = class
  private
    FAge: Integer;
    FLastname: string;
    FF_firstname: string;
  public
    constructor Create(const Firstname, Lastname: string; Age: Integer); virtual;
    property Firstname: string read FF_firstname write FF_firstname;
    property Lastname: string read FLastname write FLastname;
    property Age: Integer read FAge write FAge;
  end;
```

Na obrazec odložimo komponento TAdapterBindSource in napišemo metodo dogodka OnCreateAdapter.

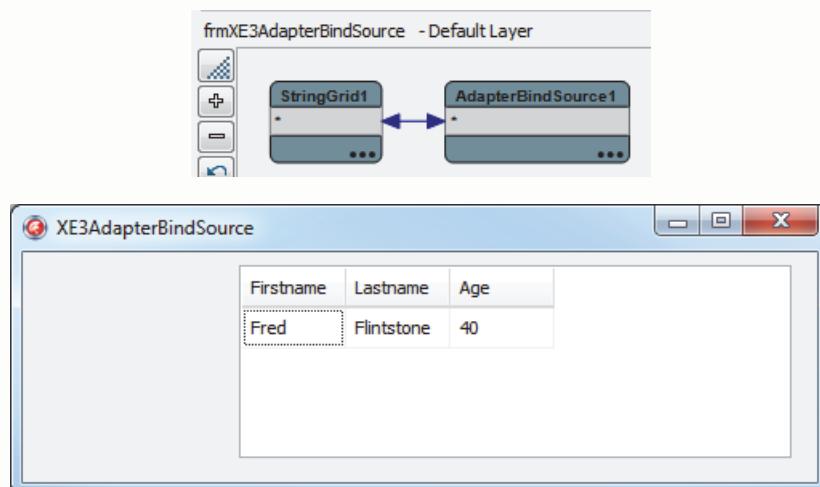
```

procedure TfrmXE3AdapterBindSource.AdapterBindSource1CreateAdapter(
  Sender: TObject; var ABindSourceAdapter: TBindSourceAdapter);
begin
  MyPerson := TPerson.Create('Fred', 'Flintstone', 40);
  ABindSourceAdapter := TObjectBindSourceAdapter<TPerson>.Create(
    Self, MyPerson, True);
end;

```

Najprej naredimo in inicializiramo primerek razred `TPerson` (tega ne moremo narediti v dogodku `FormCreate`, ker se `AdapterBindSource1.OnCreateAdapter` pokliče že pred tem). Nato naredimo primerek razred `TObjectBindSourceAdapter<TPerson>` ter mu nastavimo tri parametre. Prvi predstavlja lastnika (uporabimo kar obrazec), drugi objekt, ki ga »ovijamo« s tem adapterjem, tretji pa določa, da si adapter lasti objekt (`OwnsObject := true`). Ko bo adapter uničen, bo spotoma uničil še objekt.

Na obrazec odložimo še `TStringGrid` in prikličemo vizualni povezovalnik. Povežemo `AdapterBindSource1.*` s `StringGrid1.*` in poženemo program.



Še bolj zanimiva je možnost povezave seznama objektov z vizualnim gradnikom. V programu deklariramo še polje, ki predstavlja seznam oseb.

```
MyPeople: TObjectList<TPerson>;
```

Na obrazec postavimo še en par komponent `TAdapterBindSource` in `TStringGrid`. Povežemo `AdapterBindSource2.*` s `StringGrid2.*`. Nato napišemo metodo dogodka `OnCreateAdapter` komponente `AdapterBindSource2`.

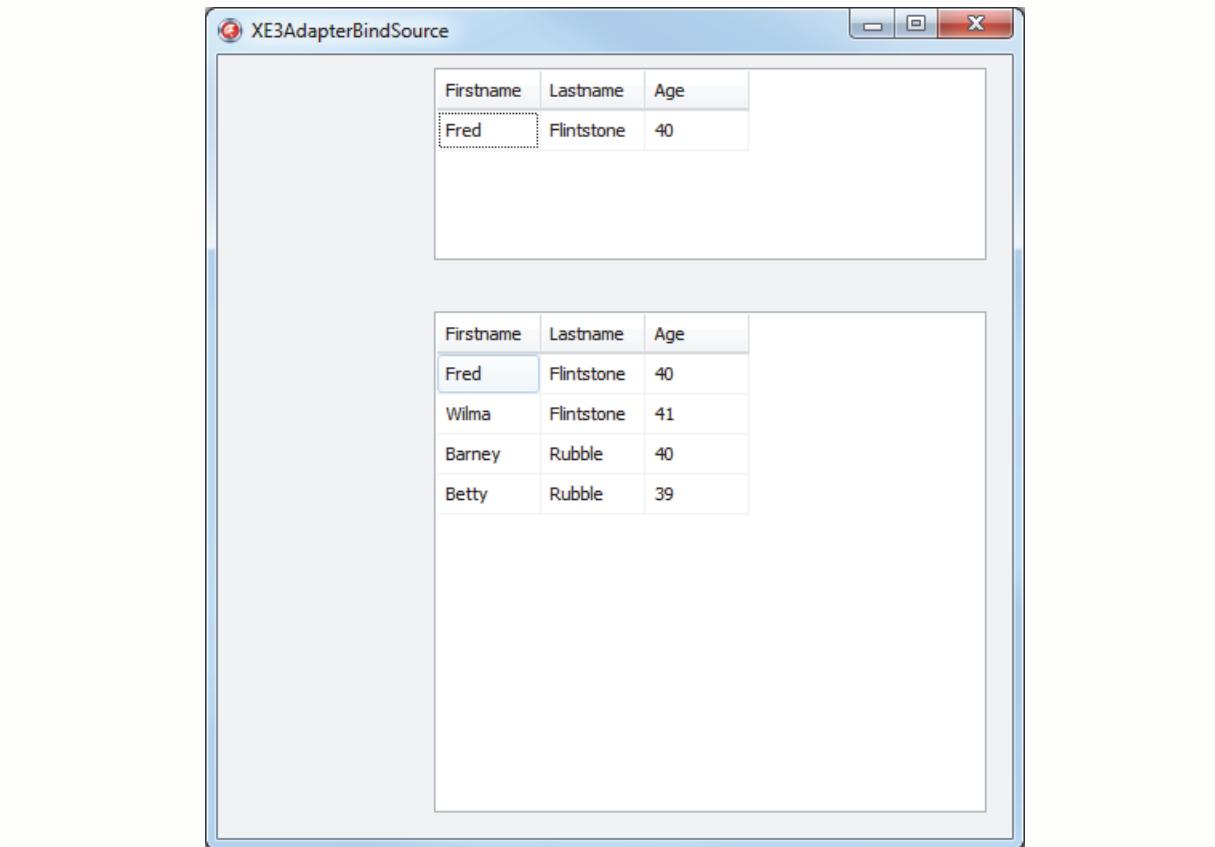
```

procedure TfrmXE3AdapterBindSource.AdapterBindSource2CreateAdapter(Sender:
  TObject;
  var ABindSourceAdapter: TBindSourceAdapter);
begin
  MyPeople := TObjectList<TPerson>.Create;
  MyPeople.Add(TPerson.Create('Fred', 'Flintstone', 40));
  MyPeople.Add(TPerson.Create('Wilma', 'Flintstone', 41));
  MyPeople.Add(TPerson.Create('Barney', 'Rubble', 40));
  MyPeople.Add(TPerson.Create('Betty', 'Rubble', 39));

  ABindSourceAdapter := TListBindSourceAdapter<TPerson>.Create(
    self, MyPeople, True);
end;

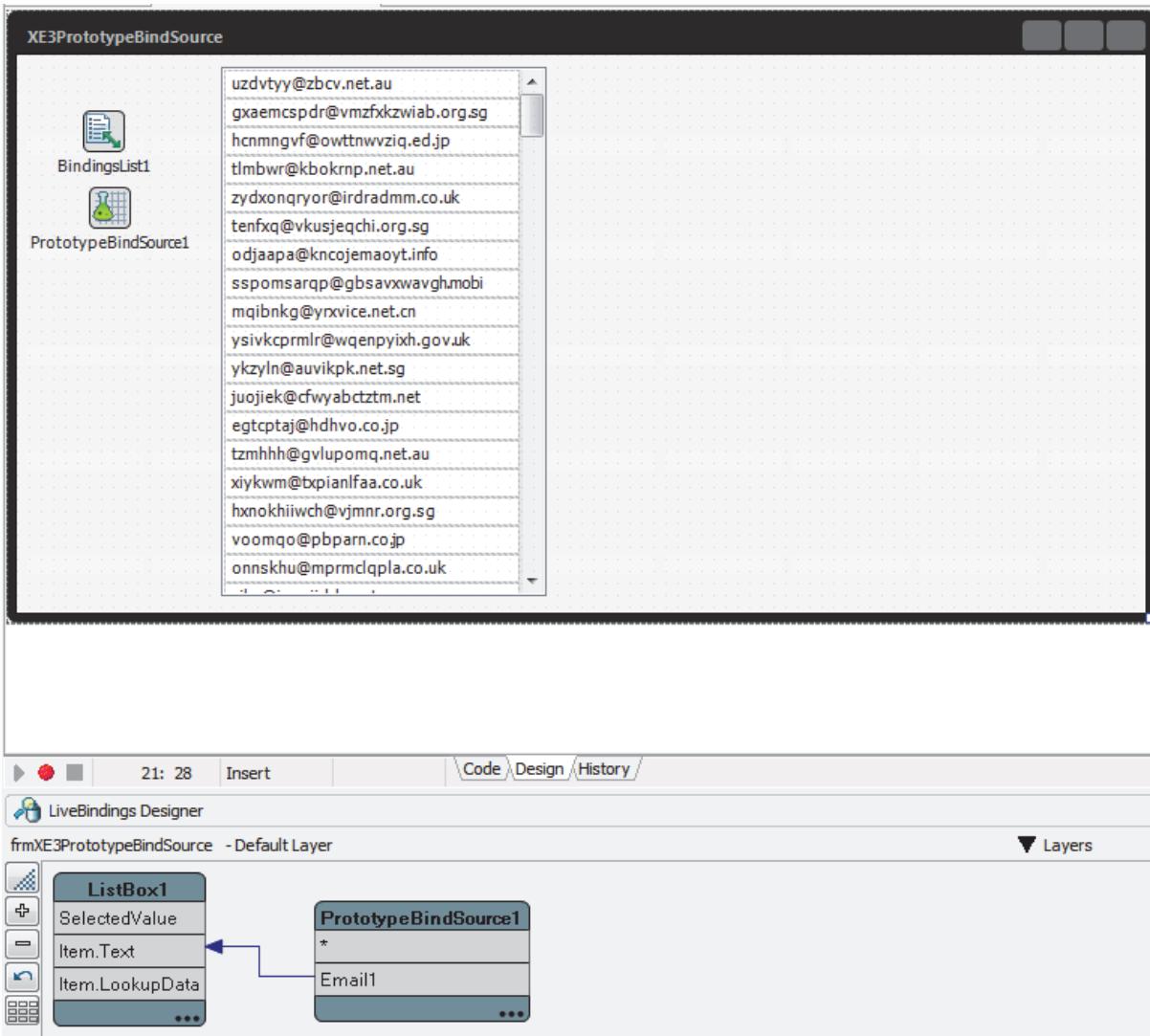
```

Tako kot v prejšnjem primeru moramo tudi tu najprej inicializirati objekt, ki ga bomo povezali. Nato na enak način kakor prej naredimo adapter, le da namesto TObjectBindSourceAdapter<TPerson> uporabimo razred TListBindSourceAdapter<TPerson>.



TPrototypeBindSource

Komponenta TPrototypeBindSource pravzaprav le združuje dve komponenti, ki ju lahko uporabimo tudi ločeno, TAdapterBindSource in TDataGeneratorAdapter. Komponenta zna generirati najrazličnejše tipe podatkov – nize, števila, datume itd. Povežemo jo lahko na poljuben izhodni gradnik in s tem simuliramo podatke iz (morda še neobstoječe) podatkovne baze. Dodamo lahko tudi lastne generatorje podatkov in z njimi razširimo osnovno komponento. Če lastne generatorje namestimo kot razširitvene pakete, jih lahko uporabimo tudi v urejevalniku. To prikazuje testni projekt XE3PrototypeBindSourceGroup.



Uporabna lastnost komponente je, da med urejanjem programa prikazuje testne podatke, med izvajanjem pa jo priključimo na notranji vir podatkov (objekte). To možnost prikazuje primer v naslednjem poglavju.

Ločitev predstavitvenega sloja in poslovnega vmesnika

Zaključimo to predstavitev z opisom praktične rabe ogrodja LiveBindings. Uporabili ga bomo za razslojitev programa na uporabniški vmesnik in poslovno logiko.

V te namene obstaja veliko vzorcev (patterns). Med najpomembnejše spadajo Model-View-Controller (MVC), Model-View-Presenter (MVP) in Model-View-ViewModel (MVVM). Več o njih si lahko preberete na Wikipedii.

- MVC: <http://en.wikipedia.org/wiki/Model-view-controller>
- MVP: <http://en.wikipedia.org/wiki/Model-view-presenter>
- MVVM: <http://en.wikipedia.org/wiki/MVVM>

Vsem je skupna ista motivacija – razdeliti program na dele, ki jih lahko avtomatsko testiramo (poslovna logika in koda, ki povezuje poslovno logiko z vizualnim vmesnikom) in dele, pri katerih je avtomatsko testiranje zapleteno (vizualni vmesnik). Razdelitve na sloje se loteva vsak vzorec po svoje; vsem je skupno le, da poslovno logiko premaknejo v sloj Model.

Tak pristop olajša programiranje za več različnih platform. Namesto da bi pisali večje dele programa na novo za vsako platformo, na novo napišemo le vmesniški del (View), ostanek (Model ter Controller/Presenter/ViewModel) pa pustimo nespremenjen.

Primernost rabe teh vzorcev je odvisna od vizualnih ogrodnih v programske orodji. Ogrodjem VCL in FireMonkey še najbolj ustreza pristop MVVM, ki si ga bomo ogledali v nadaljevanju.

Vzorec Model-View-ViewModel (MVVM)

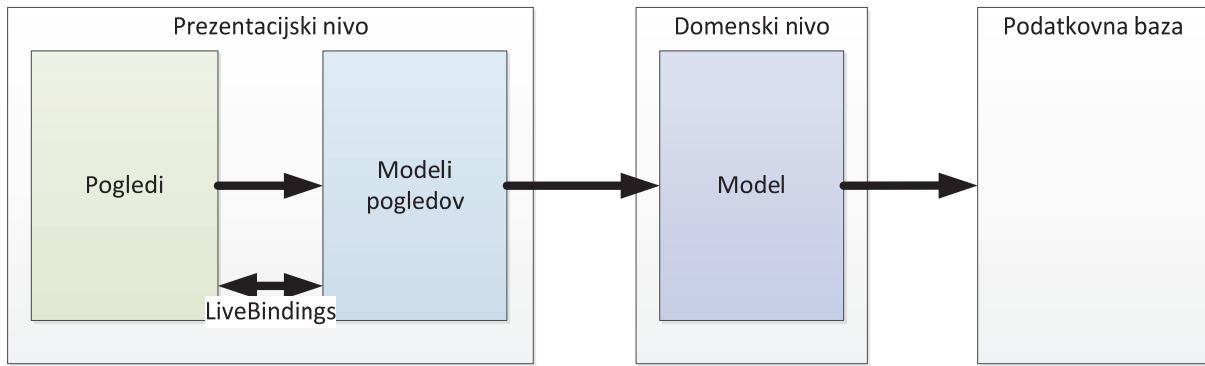
Opis vzorca MVVM temelji na predavanju »An Introduction to Model-View-View Model (MVVM) in Delphi«, ki ga je na letošnjem CodeRage 7 predstavil Malcolm Groves. Ogledate si ga lahko na naslovu <http://www.youtube.com/watch?v=Ci1HP8ZBJxk>, nekaj koristnih povezav pa je Malcolm zbral na svojem blogu, <http://www.malcolmgroves.com/blog/?p=1340>.

Bistvo vzorca MVVM je razdelitev na tri sloje.

Sloj Model vsebuje poslovno logiko. Ta sloj tudi komunicira s podatkovno bazo, če je to potrebno. Ta sloj lahko testiramo z avtomatskimi testi in ne ve nič o drugih dveh slojih.

Sloj ViewModel vsebuje enega ali več *Modelov pogledov* (ViewModel). Vsak je vmesnik med vizualnim delom (View) in Modelom. Rečemo lahko tudi, da je Model pogleda abstrakten opis uporabniškega vmesnika. Tudi komponente tega sloja lahko avtomatsko testiramo. Komponente tega sloja poznajo Model in z njim po potrebi komunicirajo, ne vedo pa nič o sloju View.

Sloj View vsebuje enega ali več *Pogledov* (View). Pogled je lahko obrazec, ali pa tudi del obrazca (TFrame, TPanel). Vsaka komponenta tega sloja komunicira z ustreznim Modelom pogleda.



Ogrodje LiveBindings lahko v tem kontekstu uporabimo na dva načina. Pri pripravi Pogledov nam TDataGeneratorAdapter ali TPrototypeBindSource omogočata ogled podatkov v uporabniškem vmesniku, žive povezave pa uporabimo za povezavo Pogledov z Modeli pogledov.

Model

V testnem programu sloj Model vsebuje le dva objekta. Prvi, TTask, opisuje eno opravilo, ki ga lahko dodamo v seznam opravil (To-do list). Implementiran je v enoti Model.Task. Drugi, TTaskList, predstavlja seznam opravil. Implementiran je v enoti Model.TaskList. Zaradi enostavnosti program podatkov ne shranjuje v bazo.

Model pogleda

Program vsebuje dva modela pogledov. Prvi, ViewModel.Main, predstavlja abstrakcijo glavnega uporabniškega vmesnika. Drugi, ViewModel.Task, predstavlja abstrakcijo okna za urejanje enega opravila.

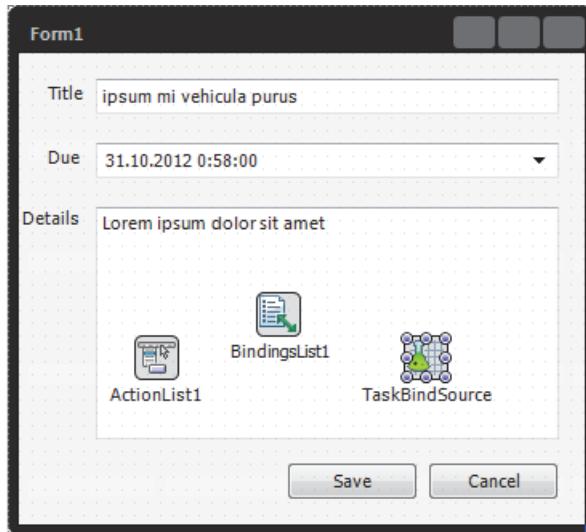
Pogled

Program vsebuje dva pogleda. Prvi je implementiran v obrazcu Views.Main, ter predstavlja glavni program, drugi pa v obrazcu View.Task in predstavlja okno za urejanje enega opravila.

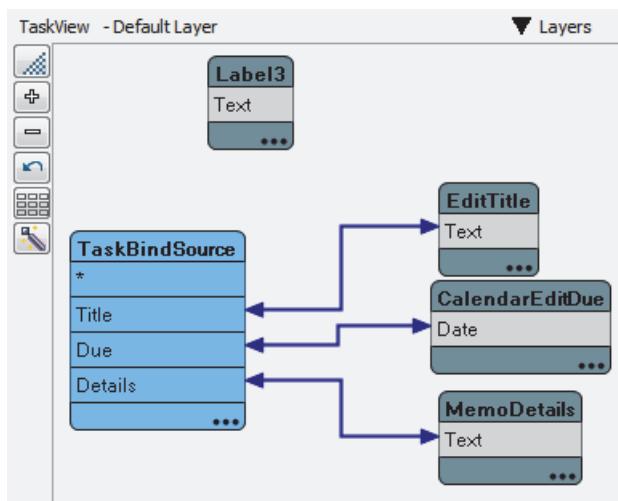
LiveBindings

Oglejmo si enostaven primer rabe ogrodja LiveBindings v testnem programu. Omejili se bomo na pogled View.Task in model pogleda ViewModel.Task.

Pogled View.Task predstavlja enostaven obrazec za urejanje opravila. Na njem najdem tudi komponento TaskBindSource: TPrototypeBindSource, ki generira testne podatke za prikaz v uporabniškem vmesniku.



Ti podatki so z vizualnimi gradniki povezani z živimi povezavami.



Komponenta `TPrototypeBindSource` ima sprogramirano metodo dogodka `OnCreateAdapter` (spomnimo se, ta komponenta združuje `TAdapterBindSource` in `TDataGeneratorAdapter`; od prvega podeduje med drugim tudi dogodek `OnCreateAdapter`). V tej metodi preusmerimo vir podatkov iz naključno generiranih na pripadajoč Model pogleda – `ViewModel.Task`.

```
procedure TTaskView.TaskBindSourceCreateAdapter(Sender: TObject;
  var ABindSourceAdapter: TBindSourceAdapter);
begin
  ABindSourceAdapter := TObjectBindSourceAdapter<TTask>.Create(
    TaskBindSource, ViewModel.Task, False);
end;
```

S tem smo vizualne gradnike dvosmerno povezali z modelom. `ViewModel.Task` je namreč lastnost objekta `TTaskViewModel`, ki spada v sloj Model pogleda, predstavlja pa objekt iz sloja Model.

Viri

FireMonkey Application Platform (dokumentacija)

http://docwiki.embarcadero.com/RADStudio/en/FireMonkey_Application_Platform

White Paper: Understanding RAD Studio LiveBindings

<http://edn.embarcadero.com/article/42076>

Using LiveBindings to Connect the UI to Objects [XE2]

<http://members.adug.org.au/2012/03/16/using-livebindings-to-connect-the-ui-to-objects/>

LiveBindings in XE3 – TBindSourceDB

<http://www.malcolmgroves.com/blog/?p=1072>

LiveBindings in XE3: TAdapterBindSource and binding to Objects

<http://www.malcolmgroves.com/blog/?p=1084>

LiveBindings in XE3: Updating Objects via an Adapter

<http://www.malcolmgroves.com/blog/?p=1186>

LiveBindings in XE3: Formatting your Fields

<http://www.malcolmgroves.com/blog/?p=1226>

LiveBindings in XE3: TPrototypeBindSource and Custom Generators

<http://www.malcolmgroves.com/blog/?p=1254>

LiveBindings Part 1 – Displaying database data in a FireMonkey application [XE2]

<http://members.adug.org.au/2011/12/29/livebindings-01/>

LiveBindings Part 2 – Displaying database data in a VCL (or FireMonkey) application

<http://members.adug.org.au/2012/01/05/livebindings-02/>

XE3 Visual LiveBindings: Samples

<http://blogs.embarcadero.com/jimtierney/2012/10/21/31944>

XE3 Visual LiveBindings: Link controls to fields

<http://blogs.embarcadero.com/jimtierney/2012/10/01/31653>

XE3 Visual LiveBindings: Actions

<http://blogs.embarcadero.com/jimtierney/2012/10/03/31721>

XE3 Visual LiveBindings: Link a list control to a field

<http://blogs.embarcadero.com/jimtierney/2012/10/14/31824>

XE3 Visual LiveBindings: TListView

<http://blogs.embarcadero.com/jimtierney/2012/10/21/31925>

XE3 Visual LiveBindings: Link a field to a lookup list

<http://blogs.embarcadero.com/jimtierney/2012/10/15/31834>

XE3 Visual LiveBindings: Lookup controls

<http://blogs.embarcadero.com/jimtierney/2012/10/20/31892>

Live Binding and Nulls

<http://esbdevlib.com/wordpress/?p=37>

TIndex - LiveBindings

<http://www.tindex.net/LiveBinding.html>

TIndex – Visual LiveBindings

<http://www.tindex.net/VisualLiveBindings.html>

Video

Introduction to the LiveBindings Designer

<http://blogs.embarcadero.com/davidi/2012/10/07/41718>

<http://www.youtube.com/watch?v=WSbulwePlkQ>

Using the LiveBindings Designer with an Object

<http://blogs.embarcadero.com/davidi/2012/10/08/41733>

<http://www.youtube.com/watch?v=DtuPZ9SiqU8>

Introduction to the LiveBindings Wizard

<http://blogs.embarcadero.com/davidi/2012/10/21/41849>

<http://www.youtube.com/watch?v=tpipscFNTGA>

Using the CustomFormat property in LiveBindings

<http://blogs.embarcadero.com/davidi/2012/10/30/41897>

<http://www.youtube.com/watch?v=86PPmORQ7kY>

Delphi Training Tutorial #77 - Visual Live Bindings

<http://www.youtube.com/watch?v=D0p0piYYgyk>

CodeRage 7 - Malcolm Groves - An Introduction to Model-View-View Model (MVVM) in Delphi

<http://www.youtube.com/watch?v=Ci1HP8ZBJxk>

<https://github.com/malcolmgroves/menialtasks/downloads> (koda)

<http://www.malcolmgroves.com/blog/?p=1340> (povezave)

Dokumentacija

LiveBindings in RAD Studio

http://docwiki.embarcadero.com/RADStudio/XE3/en/LiveBindings_in_RAD_Studio

Creating LiveBindings

http://docwiki.embarcadero.com/RADStudio/XE3/en/Creating_LiveBindings

Creating New LiveBindings in your Application

http://docwiki.embarcadero.com/RADStudio/XE3/en/New_LiveBinding

Using TPrototypeBindSource and the LiveBindings Designer

http://docwiki.embarcadero.com/RADStudio/XE3/en/Tutorial:_Using_TPrototypeBindSource_and_the_LiveBindings_Designer

LiveBindings Designer

http://docwiki.embarcadero.com/RADStudio/XE3/en/LiveBindings_Designer

LiveBindings Wizard

http://docwiki.embarcadero.com/RADStudio/XE3/en/LiveBindings_Wizard

TLinkPropertyToField

<http://docwiki.embarcadero.com/Libraries/XE3/en/Data.Bind.Components.TLinkPropertyToField>

Tutorial: Linking Controls via LiveBindings Designer

http://docwiki.embarcadero.com/RADStudio/XE3/en/Tutorial:_Linking_Controls_via_LiveBindings_Designer

Tutorial: Using TPrototypeBindSource and the LiveBindings Designer

http://docwiki.embarcadero.com/RADStudio/XE3/en/Tutorial:_Using_TPrototypeBindSource_and_the_LiveBindings_Designer

Tutorial: Linking a Control With a Field

http://docwiki.embarcadero.com/RADStudio/XE3/en/Linking_a_Control_with_a_Field

Tutorial: Linking a Control With a Component Property

http://docwiki.embarcadero.com/RADStudio/XE3/en/Linking_a_Control_with_a_Component_Property

Tutorial: Linking a Property of a Component With a Control

http://docwiki.embarcadero.com/RADStudio/XE3/en/Linking_a_Property_of_a_Component_with_a_Control

Tutorial: Linking a Property of a Component With a Field

http://docwiki.embarcadero.com/RADStudio/XE3/en/Linking_a_Property_of_a_Component_with_a_Field

Tutorial: Creating a Data Source

[http://docwiki.embarcadero.com/RADStudio/XE3/en/Creating_a_Data_Source_\(LiveBindings_Wizard\)](http://docwiki.embarcadero.com/RADStudio/XE3/en/Creating_a_Data_Source_(LiveBindings_Wizard))

Tutorial: Using LiveBinding in VCL Applications (exemplifies VCL Application property binding)

http://docwiki.embarcadero.com/RADStudio/XE3/en/Tutorial:_Using_LiveBinding_in_VCL_Applications

Tutorial: Using LiveBindings in FireMonkey Applications (exemplifies FireMonkey HD Application property binding)

http://docwiki.embarcadero.com/RADStudio/XE3/en/Tutorial:_Using_LiveBindings_in_FireMonkey_Applications

Tutorial: Using LiveBinding to Create an Application Without Code (exemplifies a FireMonkey HD Application built using LiveBindings)

http://docwiki.embarcadero.com/RADStudio/XE3/en/Tutorial:_Using_LiveBindings_in_FireMonkey_Applications

Tutorial: Using LiveBinding Programatically (only suitable for Console Applications or manual expression editing)

http://docwiki.embarcadero.com/RADStudio/XE3/en/Tutorial:_Using_LiveBinding_Programatically