V1.1

embarcadero
reseller partner

Primož Gabrijelčič
http://thedelphigeek.com

MARAND
Napredna računalniška hiša

Kazalo

Vsi programi, omenjeni v tem dokumentu, so na voljo na naslovu http://17slon.com/EA/EA-AndroidXE5.zip.

Uvod

V različici XE5 programsko okolje RAD Studio XE5 (v jeziku Delphi) in njegova podizvedba Delphi XE5 omogočata izdelavo programov za operacijski sistem Android. V naslednji različici pa lahko pričakujemo podporo za Android tudi v jeziku C++ in okolju C++Builder (že v različici XE5 z nadgradnjo Update 2 bo v jeziku C++ možno programiranje za iOS).

Za razvoj mobilnih programov potrebujete RAD Studio XE5 ali Delphi XE5 različice Enterprise ali boljše oziroma različico Professional z dodatko Mobile Add-On Pack. Za delo s podatkovnimi bazami na mobilni napravi potrebujete različico Enterprise ali boljšo oziroma različico Professional in dodatek FireDAC Client/Server Add-On Pack.

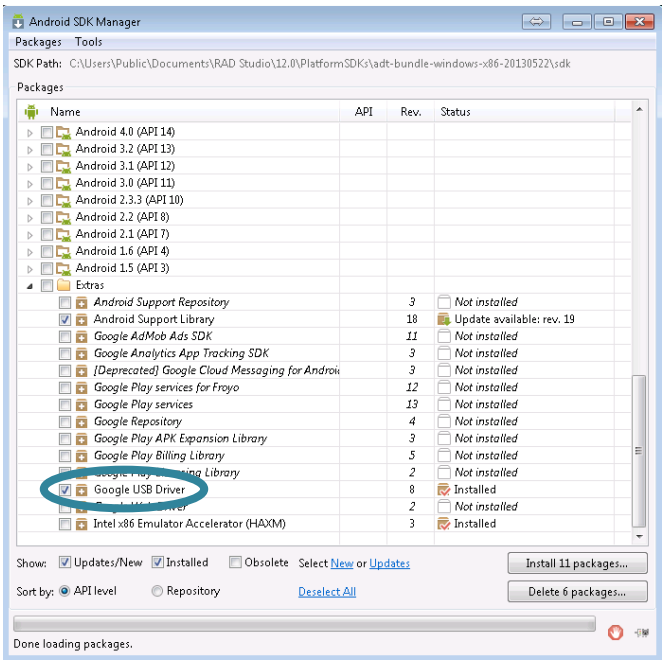
Z Delphijem izdelani programi bodo tekli na Androidnih različicah Gingerbread (2.3.3 – 2.3.7), Ice Cream Sandwich (4.0.3 – 4.0.4) in Jelly Bean (4.1.x, 4.2.x, 4.3.x). Različica Honeycomb (3.x) ni podprta. Naprave potrebujejo procesor ARMv7 s podporo nabora ukazov NEON. Podprti so tako telefoni kot tudi tablični računalniki. Združljivost lahko preverite s prostim programom SysCheck (naslov najdete med viri na koncu skripte).

Programe, ki jih boste izdelali z Delphi XE5, lahko brez težav distribuirate v tržnici Google Play ali pa jih na naprave nameščate neposredno iz lastnih strežnikov.

Namestitev

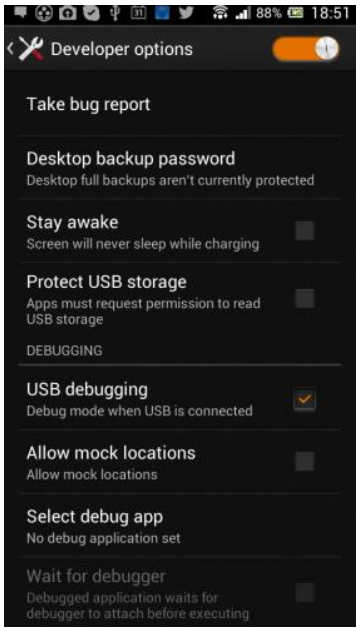
Priporočamo, da namestite RAD Studio/Delphi XE5 s privzetimi nastavitvami. Namestitveni program bo namestil tudi razvojno okolje Android SDK. V primeru, da je ta že nameščen na računalnik, lahko njegovo namestitev v RAD Studio/Delphi XE5 preskočite.

Za preizkušanje delovanja programov na napravah (kar toplo priporočamo) boste potrebovali kabel USB in gonilnike za napravo, ki podpirajo razhroščevanje z Androidovim orodjem *adb*. V večini primerov jih boste morali prenesti neposredno od izdelovalca telefona/tablice, saj privzeto nameščeni gonilniki razhroščevanja pogosto ne podpirajo. Nekaj povezav na vire gonilnikov smo zbrali na koncu skripte.



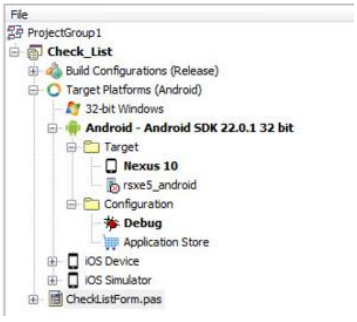
Če imate Googlov telefon/tablico (Nexus), lahko poženete *Android SDK Manager* (ikona se privzeto namesti v skupino *RAD Studio XE5*) ter prižgete možnost *Google USB Driver*.

Z omenjenim orodjem lahko namestite različne različice razvojnega orodja ter pri prevajanju v Delphiju uporabite katerokoli od njih.



Na napravi morate nato vključiti razvijalske možnosti (*Developer Options*) ter v njih možnost razhroščevanja s povezavo USB (*USB Debugging*). V Androidih do (vključno) 4.1 je ta možnost vedno dostopna, v različici 4.2 in novejših pa je skrita in jo morate vključiti z drobnim trikom:

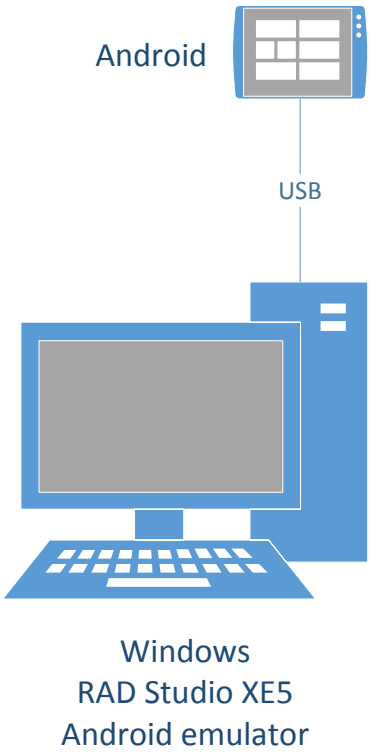
- Odprite *Settings*, *About Device*.
- Oddrsajte do *Build Number*.
- Sedemkrat(!) tapnite to izbiro.
- Pojavilo se bo sporočilo *»Developer mode has been enabled«*.
- V nastavitvah (*Settings, System*) se bo pojavila možnost *Developer options*.



Ko so gonilniki pravilno nameščeni, se bo priključena naprava pojavila v seznamu *Target platforms* mobilnega projekta. Biti mora tudi vidna v seznamu, ki ga vrne ukaz *adb devices* (orodje *adb* je privzeto nameščeno v mapi *C:\Users\Public\Documents\RAD Studio\12.0\PlatformSDKs\adt-bundle-windows-x86-20130522\sdk\platform-tools*).

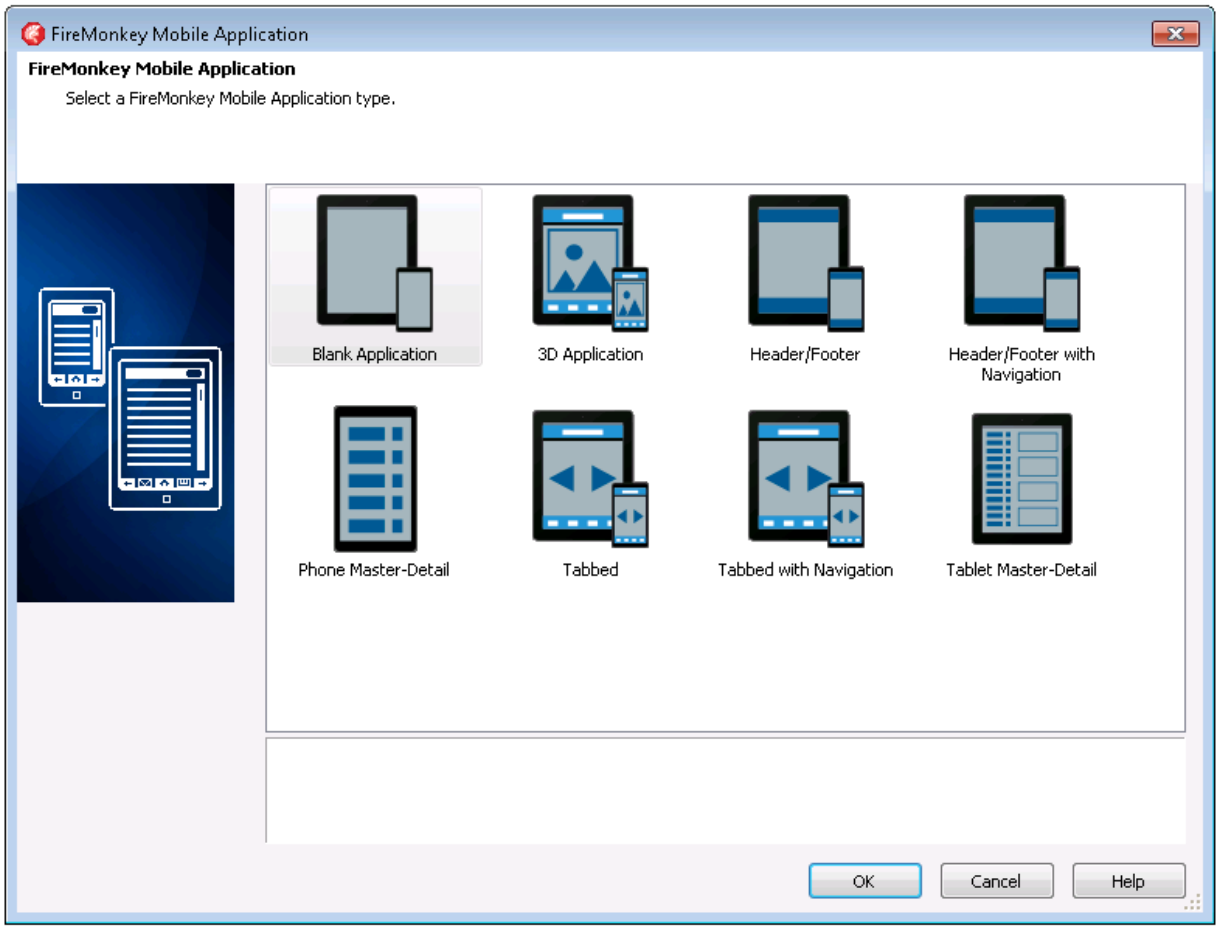
Programe lahko poganjate tudi v emulatorju, a to možnost zelo odsvetujemo – androidni emulator je namreč strašno počasen program. Postopek izdelave emulatorja in priprave za rabo v Delphiju/RAD Studiu je dobro opisan v spletni dokumentaciji (viri na koncu knjige). En emulator (temelječ na Nexusu 7) bo namestil že namestitveni program.

Postopek namestitve Delphija / RAD Studia je dobro opisan na spletišču *http://docwiki.embarcadero.com* (glej *Viri*).

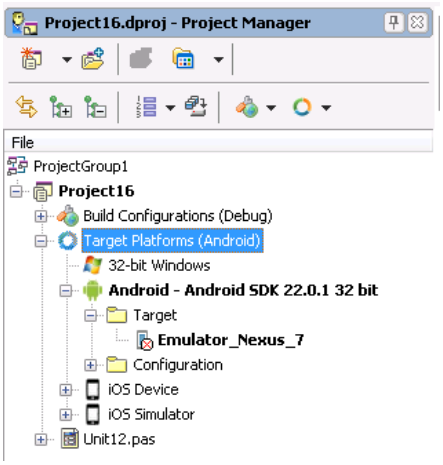


Razvoj

Programi za Android so vedno napisani s platformo FireMonkey, ki jo v RAD Studiu XE5 srečamo že v tretji reinkarnaciji. Projekt začnete z izbiro možnosti *File, New, FireMonkey Mobile Application.* Na voljo imate več predlog, ki vam bodo olajšale začetno raziskovanje okolja. Ko boste te možnosti prerasli, pa lahko nove programe začnete na podlagi predloge *Blank Application*.



Med razvojem programa lahko poljubno spreminjate ciljne naprave in program testirate v različnih okoljih. Vsa podprta okolja so zbrana v oknu *Project Manager* v veji *Target Platforms*. Tam boste našli podveje za vse nameščene različice *Android SDK*, znotraj vsake pa seznam trenutno priključenih naprav in nameščenih emulatorjev. Videli boste, da lahko program poženete tudi na napravi z operacijskim sistemom iOS ali v simulatorju take naprave, za kar pa potrebujete računalnik z operacijskim sistemom OS X.



Mobilne programe lahko poženete tudi na računalniku, na katerem razvijate program, če kot ciljno platformo izberete *32-bit Windows*. V tem primeru bo program pognan v vizualno okrnjeni različici (in seveda ne bo imel dostopa do posebne strojne opreme – kompasa, dajalnika položaja, merilca pospeškov, kamere itd), zato je tak način primeren le za hitre teste.

Ko je razvojni sistem pravilno postavljen, je delo z napravo silno enostavno – izberete ciljno napravo, pritisnete F9 in program bo pognan na njej. Omogočeno je tudi neposredno razhročevanje v razvojnem okolju.

Nasvet: Razhroščevalnik še ni popolnoma stabilen in včasih povzroči, da se orodje adb *neha pravilno obnašati. Takrat si lahko pomagate z ukazoma* adb kill-server*,* adb start-server*, ki ponovno zaženeta strežnik* adb.

Za prenos v trgovino Google Play boste morali nastaviti še nekaj lastnosti projekta v *Project, Project Options*. V veji *Uses Permissions* nastavite pravice, ki jih bo vaš program zahteval ob namestitvi. V veji *Application* nastavite ikone in začetno sliko (splash screen) ter orientacije, ki jih program podpira. V veji *Provisioning* pa nastavite ključe in gesla za trgovino Play.

Če program potrebuje dodatne datoteke (slike, tekstovne datoteke, knjižnico MIDAS etc), jih nastavite v menuju *Project, Deployment.*

Nasvet: Med razvojem vam bo koristilo orodje, ki sliko zaslona telefona preslika na računalnik ali orodje, ki omogoča oddaljeni dostop do telefona/tablice. Nekaj koristnih povezav smo zbrali v Virih.

NextGen Delphi

Kodo za mobilne naprave (tako iOS kakor tudi Android) prevaja novi prevajalnik, ki ga označujejo kot prevajalnik naslednje generacije (*NextGen*). Ta ima v primerjavi s prevajalnikom za Windows/OS X kar nekaj sprememb.

- Podprt je le še en tip nizov – *UnicodeString* oziroma *string*. Vsi ostali tipi (AnsiString, UTF8String, WideString, OpenString, ShortString ...) so ukinjeni.
- Stringi so indeksirani od 0 naprej, ne od 1. (To lahko spremenite z direktivo {$ZEROBASEDSTRINGS OFF}.) Standardne funkcije (Pos, Copy, Delete ...) zaradi združljivosti še vedno delujejo z indeksi od 1 dalje (torej bo *Copy(s, 1, 2)* vrnil prva dva znaka niza *s*), dodane pa so funkcije razreda TStringHelper, ki jih uporabimo v obliki *niz.funkcija()* in uporabljajo indekse od 0 naprej (funkcija *s.Substring(0, 2)* vrne prva dva znaka niza *s*).
- V prevajalniku za Windows/OS X so nizi privzeto indeksirani od 1 dalje, to pa lahko spremenite z direktivo {$ZEROBASEDSTRINGS ON}.
- Tipa *pointer* ni več, uporabljamo lahko le še kazalce na objekte (na primer *^TObject*).
- Zbirnika ne morete več pisati sredi pascalskih metod (to so že pred časom ukinili za Win64 prevajalnik).
- Objektov ni več treba sproščati (Free, FreeAndNil), lahko pa to še vedno počnete. Vsi objekti vsebujejo števec referenc (ARC; Automatic Reference Count – to so doslej poznali vmesniki (interface)) in ko števec pade na 0, prevajalnik samodejno sprosti objekt. Spodnja koda je po novem popolnoma legalna in ne povzroči izgubljenega pomnilnika.

```
procedure TMySimpleClass.CreateOnly;
var
  MyObj: TMySimpleClass;
begin
  MyObj := TMySimpleClass.Create;
  MyObj.DoSomething;
end
```

- Cikle kazalcev (objekt A vsebuje naslov objekta B, ta pa vsebuje naslov objekta A) prekinete z novim atributom *[weak]*, ki določi, da deklaracija, ki sledi, ne predstavlja lastništva objekta in zato ne poveča števca referenc.
- Operatorje lahko definiramo tudi na razredih, ne samo na recordih.

Te spremembe veljajo le **mobilne** programe, ne veljajo pa za **namizne** programe za Windows in OS X, kjer kodo izdeluje stari prevajalnik.

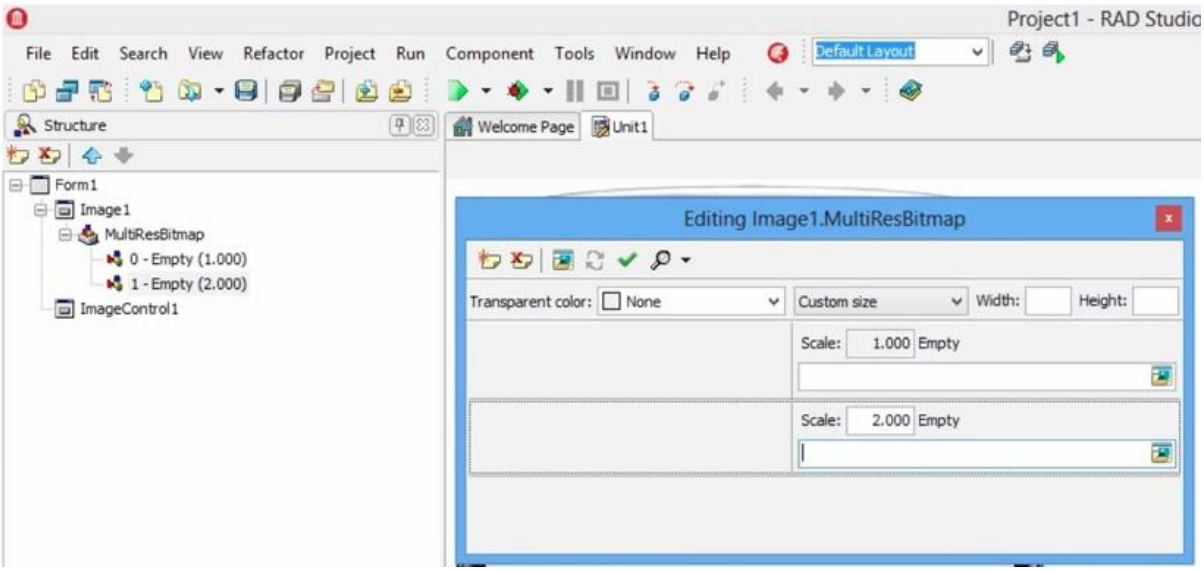
FireMonkey 3

Razvoj Androidnih programov v RAD Studiu je vedno podprt s platformo FireMonkey, ki se v XE5 pojavlja že v tretji različici. O platformi FireMonkey smo pripravili že več predavanj, ki so na voljo v spletu, od kjer lahko prenesete tudi skripto (glej Viri), zato se bomo tu omejili le na novosti in lastnosti, pomembne za razvoj mobilnih programov.

V različici 3 mobilne naprave vedno uporabljajo kanvas (površino za risanje), ki jo poganja grafični soprocesor (GPU canvas). Ker ta ne deluje z vektorskimi slogi iz različice 1, imate na voljo le še bitne sloge, ki jih je uvedla različica 2.

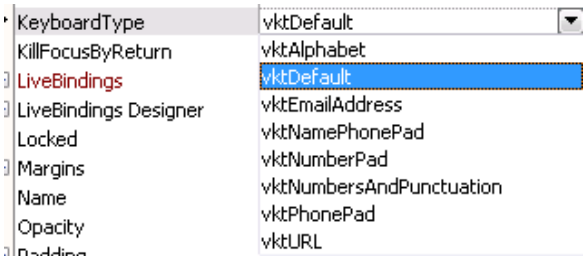
Zaradi velikih razlik v zaslonih mobilnih naprav (tako v skupni ločljivosti kakor tudi v številu pik na palec), je FireMonkey 3 vpeljal koncept bitnih slik z več ločljivostmi. V bistvu gre za zbirko bitnih slik različnih dimenzij, kjer vsaki sliki predpišemo, za katero gostoto prikaza pik (*Scale* na spodnji sliki) se bo bitna slika uporabila. Če gostote prikaza, ki jo uporablja ciljna naprava, ni v seznamu, se bo uporabil najboljši približek.

Z večločljivostnimi bitnimi slikami ne delamo v pikah, temveč v logičnih enotah (LU), kjer je 1 LU lahko enak eni piki, ali pa tudi ne. Razmerje 1 pika = 1 LU velja za običajne zaslone na napravah iPhone/iPad ter na starejših Androidih. Zasloni Retina na novejših iPhonih/iPadih uporabljajo razmerje 1 LU = 2 piki (*Scale = 2*), pri Androidih pa je stanje zelo raznoliko, saj najdemo razmerja 1:1, 1,33:1, 1,5:1, 2:1 in tudi 3:1. Trenutno velja priporočilo, da v program vključite bitne slike v razmerjih 1:1, 1,5:1 in 2:1, kar bo zadoščalo za večino naprav.



V FireMonkey je vključena množica gradnikov (controls), ki si jih lahko ogledate v programu *MobileControls* (najdete ga med drugimi primeri, ki se namestijo v mapo *C:\Users\Public\Documents\RAD Studio\12.0\Samples*). Nekateri gradniki so izvedeni v kodi in se izrišejo glede na trenutno izbrani slog, spet drugi pa predstavljajo le vmesnik do sistemskih gradnikov (tako kot delujejo gradniki v platformi VCL) in jih izriše operacijski sistem. Takšni so na primer *TComboBox* (izbiranje iz seznama), *TEdit* in *TMemo* (vnos besedila), *TCalendarEditor* (vnos datuma) in še kakšnega bi se našlo.

Ker sta *TEdit* in *TMemo* sistemska gradnika, podpirata vse nameščene tipkovnice, preverjanje črkovanja, kopiranje in lepljenje ter druge sistemske funkcije. Določite jima lahko tudi vrsto tipkovnice ki bo uporabljena, ko bo gradnik aktiven.



Za izdelavo večstranskih vmesnikov običajno uporabljamo *TTabControl*. V njem lahko jezičke postavimo na vrh, dno, jih skrijemo ali pa jih predstavimo s pikicami na dnu zaslona (lastnost *TabPosition*). Prek gradnika *TGestureManager* lahko *TTabControl* povežemo s kretnjama *left* in *right* s standardno akcijo *TChangeTabAction* in tako brez pisanja kode dosežemo enostavno premikanje po jezičkih.

Za prikaz seznamov uporabljamo *TListBox* ali *TListView*. Prvi je bolj prilagodljiv, saj lahko v vsak element seznama postavimo poljubne gradnike. Je pa zaradi tega tudi počasnejši in primeren za prikaz manjšega števila podatkov. Novejši *TListView* je hitrejši in bolje prikazuje večjo količino podatkov, je pa manj prilagodljiv, saj morajo biti vsi elementi seznama enako oblikovani. *TListView* ima vgrajeno tudi možnost filtriranja in brisanja elementov.

Dialogi

V mobilni različici FireMonkeya lahko za prikaz sporočil in enostaven vnos podatkov uporabljate iste funkcije kakor v namizni različici – *ShowMessage*, *MessageBox*, *InputQuery*.

```
procedure TfrmSecondaryForm.Button4Click(Sender: TObject);
begin
  MessageDlg('Torej?', TMsgDlgType.mtConfirmation,
    mbYesNoCancel, -1, TMsgDlgBtn.mbYes);
end;

procedure TfrmSecondaryForm.Button5Click(Sender: TObject);
var
  s: string;
begin
  if InputQuery('Vnos podatkov', 'Vrednost', s) then
    ShowMessage('Vnesli ste: ' + s);
end;
```

Na voljo so vam tudi modalni obrazci, le da se morate zavedati treh dejstev. Prvič, obrazci so vedno raztegnjeni čez cel zaslon. Drugič, uporabiti morate posebno obliko klica *ShowModal*, ki deluje *asinhrono*. O rezultatu modalnega obrazca boste obveščeni šele s klicem funkcije, ki jo posredujete klicu *ShowModal*. Tretjič, modalni obrazec se ne bo zaprl samodejno, temveč ga morate zapreti sami.

```
procedure TfrmShowModal.Button1Click(Sender: TObject);
begin
//  frmSecondaryForm.ShowModal;

  frmSecondaryForm.ShowModal(ProcessResult);
end;

procedure TfrmShowModal.ProcessResult(res: TModalResult);
begin
  frmSecondaryForm.Close;

  if res = mrOK then
    ShowMessage('OK')
  else
    ShowMessage('Cancel');
end;
```

Prilagodljivo oblikovanje

Program, ki se prilagaja različnim velikostim zaslonov (tako imenovani *responsive design*) lahko izdelamo na več načinov.

Najprej si lahko pomagamo z lastnostmi za samodejno raztezanje (*Align*) in »pripenjanje« na rob zaslona (*Anchors*). Poleg tega lahko gradnike razpostavimo v enega od gradnikov za samodejno razvrščanje (*TFlowLayout*, *TGridLayout*) ali v splošen »vsebnik« *TLayout*.

Če to ne zadošča, lahko izdelamo več form ter vsem določimo isto lastnost *FormFamily*. Nato to lastnost registriramo v glavnem programu z ukazom:

```
Application.RegisterFormFamily(ime_družine, [TForma1, TForma2, …]);
```

Kadar v vodoravnem (*landscape*) in navpičnem (*portrait*) načinu potrebujemo drugačen razpored gradnikov, lahko zgradimo dve formi in nato v dogodku *OnResize* vsake preverimo, če moramo prikazati drugo formo. Tako delovanje demonstrira primer *Forms*, ki ga dobite z RAD Studiom.

```
procedure TLSForm.FormResize(Sender: TObject);
begin
  if (Height > Width) and Visible and Assigned(PForm) then
    PForm.Show;
end;
```

Lahko posežete tudi v glavni program (.dpr) in med inicializacijo programa preverite lastnosti naprave ter določite glavno formo.

```
{$IFDEF IOS}
function IsTablet: Boolean;
begin
  Result := TUIDevice.Wrap(TUIDevice.OCClass.currentDevice).userInterfaceIdiom =
              UIUserInterfaceIdiomPad;
end;
{$ENDIF}
{$IFDEF ANDROID}
function IsTablet: Boolean;
begin
  Result := (MainActivity.getResources.getConfiguration.screenLayout and
               TJConfiguration.JavaClass.SCREENLAYOUT_SIZE_MASK)
            >= TJConfiguration.JavaClass.SCREENLAYOUT_SIZE_LARGE;
end;
{$ENDIF}

begin
  Application.Initialize;
  if IsTablet then
    Application.CreateForm(TTabletMainForm, TabletMainForm)
  else
    Application.CreateForm(TPhoneMainForm, PhoneMainForm);

  Application.Run;
end.
```

V skrajnem primeru lahko v lastnostih programa določite, da bo uporabljal le eno orientacijo naprave, a v splošnem je to slaba rešitev, ki je uporabniki ne marajo. Še najmanj moteča je omejitev na navpično orientacijo pri telefonih, tablice pa radi uporabljamo v obeh možnih položajih.

Tipkovnica

Delo s tipkovnico je na splošno zelo enostavno, saj večino dela postori platforma FireMonkey. Vseeno pa je koristno poznati nekaj trikov.

Standardna tipka »nazaj« na ohišju Androidnih telefonov in tablic je v sistemu predstavljena s kodo *vkHardwareBack*. Če bi radi preprečili privzeto obnašanje programa (tipično se bo ob pritisku na to tipko program zaprl in uporabnik se bo znašel pred namizjem), jo lahko ulovimo v dogodku *FormKeyUp.*

```
procedure TForm1.FormKeyUp(Sender: TObject; var Key: Word;
  var KeyChar: Char; Shift: TShiftState);
begin
  if Key = vkHardwareBack then
    Key := 0;
end;
```

Ko uporabnik klikne v vnosno polje, se na zaslonu pojavi navidezna tipkovnica (če naprava nima priključene fizične tipkovnice). Ob tem lahko tipkovnica vnosno polje prekrije. Upali bi, da bo FireMonkey samodejno premaknil vnosno polje na vidno mesto, pa ni tako. To moramo narediti sami. Postopek zahteva kar nekaj dela in je prikazan v primeru *ScrollableForm*. Narediti moramo sledeče:

- Na formo postavimo gradnik *TVertScrollBox*, vanj pa ostale gradnike.
- Napišemo odzivnik za dogodek *OnVirtualKeyboardShown*, v katerem:
  - Vprašamo sistem, kateri del zaslona je pokrila tipkovnica.
  - Premaknemo *TVertScrollBox*, da bo aktivni gradnik postal viden.
- Napišemo odzivnik za dogodek *OnVirtualKeyboardHidden*, v katerem premaknemo *TVertScrollBox* na začetno pozicijo.

Sporočila

Z gradnikom *TNotificationCenter* pripravimo sporočila, ki se pojavijo v centru za obveščanje in (po želji) pripeljejo uporabnika v naš program, če klikne sporočilo. Sporočilo se lahko pojavi takoj, ali pa ob želenem času.

V obeh primerih najprej izdelamo sporočilo s klicem *NotificationCenter.CreateNotification*, kar nam vrne objekt *TNotification*. Temu objektu nastavimo vsaj naslov (Name) in telo sporočila (AlertBody), če želimo sporočilo prikazati z zamikom, pa še lastnost FireDate (TDateTime, ko naj se sporočilo prikaže). Za takojšnji prikaz pokličemo *NotificationCenter.PresentNotification*, za zakasnjen prikaz pa *NotificationCenter.ScheduleNotification*.

```
procedure TNotificationsForm.btnSendScheduledNotificationClick(Sender: TObject);
var
  Notification: TNotification;
begin
  { verify if the service is actually supported }
  if NotificationC.Supported then begin
    Notification := NotificationC.CreateNotification;
    try
      Notification.Name := 'MyNotification';
      Notification.AlertBody := 'Delphi for Mobile is here!';

      { Fired in 10 second }
      Notification.FireDate := Now + EncodeTime(0,0,10,0);

      { Send notification in Notification Center }
      NotificationC.ScheduleNotification(Notification);
    finally
      Notification.DisposeOf;
    end;
  end
end;
```

Obvestila lahko tudi prekličemo, če pokličemo *NotificationCenter.CancelNotification*.

Pošiljanje sporočil je prikazano v primeru *SendCancelNotification*.

Če želimo biti obveščeni, ko uporabnik klikne sporočilo, moramo zgoraj omenjenemu objektu nastaviti še lastnosti *AlertAction* in *Number* ter postaviti *HasAction* na True. Prav tako moramo *NotificationCentru* nastaviti odzivnik za dogodek *OnReceiveLevelNotification*. Ko bo uporabnik kliknil sporočilo, se bo sprožil ta dogodek, v njem pa lahko dostopamo do notifikacijskega objekta in njegovih lastnosti ter se tako odločimo, kakšno akcijo bomo izvedli.

Z isto komponento lahko nastavimo tudi številko »bedža« - majhno številkico, ki se pojavi nad ikono programa na namizju in jo (na primer) poštni programi uporabljajo za prikaz števila neprebranih sporočil. To naredimo tako, da nastavimo lastnost *ApplicationIconBadgeNumber*. Raba je prikazana v primeru *SettingResettingBadgeNumber.* [Žal Android »bedžov« še ne podpira in koda na Androidu ne naredi nič. Deluje pa na napravah iOS.]

Deljenje podatkov

Pošiljanje podatkov v druge programe je izredno enostavno. Pognati moramo standardno akcijo *TShowShareSheetAction* (lahko jo sprožimo v kodi, ali pa jo pripnemo na gumb) ter napisati odzivnik za dogodek *OnBeforeExecute*, v katerem nastavimo podatke, ki jih želimo deliti z drugimi. Delimo lahko dva tipa podatkov – besedilo (zapišemo ga v lastnost *TextMessage*) in bitno sliko (priredimo jo lastnosti *Bitmap*).

Primer *ShareSheet* prikazuje, kako z mobilno aplikacijo zajamemo sliko (o tem bomo povedali več v razdelku *Standardne akcije*) ter jo nato pošljemo v drug program z akcijo *TShowShareSheetAction*. Podatke pripravimo v metodi *ShowShareSheetAction1BeforeExecute*.

```
procedure TShareSheetForm.ShowShareSheetAction1BeforeExecute(Sender: TObject);
begin
  { show the share sheet }
  ShowShareSheetAction1.Bitmap.Assign(imgCameraPicture.Bitmap);
end;
```

Deljenje podatkov lahko uporabimo tudi za tiskanje – podatke pripravimo v tekstovni ali bitni obliki in prikličemo »*share sheet*«, kjer uporabnik kot cilj izbere Googlovo storitev *Cloud Printing.*

Zvok in slika

Avdiovizualne naprave v platformi FireMonkey krmilimo z razredom *TCaptureDeviceManager.* Ta nam vrne vse A/V naprave (lastnost *Devices*), naprave nekega tipa (*GetDevicesByMediaType*) ali pa privzeto avdio ali video napravo (*DefaultAudioCaptureDevice*/*DefaultVideoCaptureDevice*).

Avdiovizualne naprave podpirajo snemanje neposredno v datoteko. Napravi nastavimo lastnost *FileName* ter sprožimo snemanje s klicem *StartCapture*. Snemanje zaustavimo s klicem *StopCapture*. Zajem zvoka je prikazan v primeru *AudioRecPlay*.

Med zajemom videa lahko dobimo dostop do vsake sličice z dogodkom *OnSampleBufferReady*. To lahko izkoristimo za prikaz videa na zaslonu.

Kamera (*TVideoCaptureDevice*) podpira tudi dostop do bliskavice (*FlashMode, HasFlash*) in svetleče diode (*TorchMode, HasTorch*). S slednjo lahko v eni vrstici sprogramiramo aplikacijo, ki oponaša baterijsko svetilko.

Sliko in zvok lahko predvajamo z gradnikom *TMediaPlayer* ali *TMediaPlayerControl*.

Senzorji

FireMonkey omogoča dostop do vseh tipal v mobilni napravi. Beremo lahko podatke iz merilca pospeškov, žiroskopa, kompasa, merilca bližine človeka, merilca svetlobe, dajalnika položaja in drugih tipal.

Podatke o lokaciji nam vrača gradnik *TLocationSensor*. Nastavimo mu lahko lastnosti *Accuracy* (želena natančnost v metrih) in *Distance* (najmanjša zanimiva razdalja od zadnje meritve). Slednja lastnost je še posebej zanimiva – če jo denimo nastavimo na 100 (metrov), potem za premike, manjše od 100 metrov, komponenta ne bo vračala spremembe položaja (tako imenovani *boxing*). Nov položaj dobimo v dogodku *OnLocationChanged*. Raba je prikazana v primerih *LocationDemoProject* in *Location*.

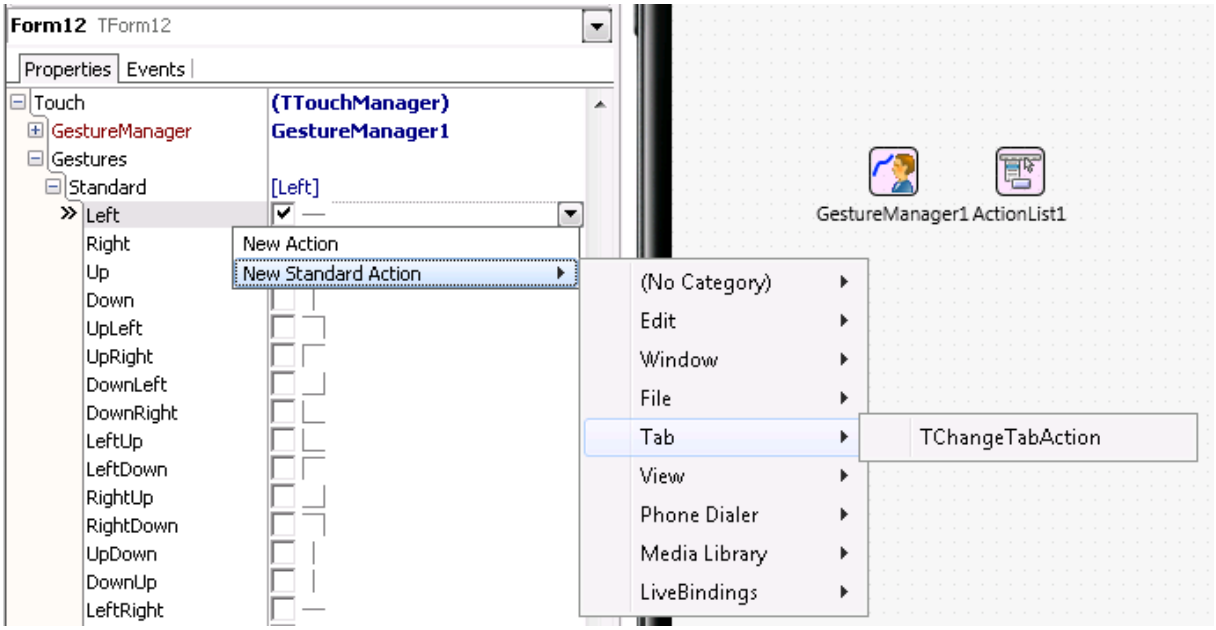
Stanje merilnika pospeška vrača gradnik *TMotionSensor*. Ko je aktiven, vrača podatke v dogodku *OnDataChanged*. Lahko pa ta dogodek ignoriramo (kliče se namreč vsaj nekaj desetkrat na sekundo) in podatke preprosto beremo s časovnikom (*TTimer*). Gradnik zna vrniti stanje merilnika pospeškov (*Sensor.AccelerationX(YZ)*) in žiroskopa (*Sensor.AngleAccelX(YZ)*), od njega pa lahko dobimo tudi hitrost premikanja naprave (*Sensor.Speed*). Raba je prikazana v primeru *Accelerometer*. Drugačen način dostopa do žiroskopa (s spodaj opisanim *TSensorManager*) je prikazana v primeru *Gyroscope*.

Podatke kompasa vrača gradnik *TOrientationSensor*. Raba je prikazana v primeru *OrientationSensor*.

Do drugih tipal pridemo z razredom *TSensorManager*. Ta nam lahko vrne vsa podprta tipala (lastnost *Sensor*), ali pa tipala nekega tipa (*GetSensorByCategory*). Vsako tipalo podpira vsaj lastnosti razreda *TCustomSensor*, nekatera tipala pa znajo vračati še več podatkov. Vsi razredi za delo s tipali so shranjeni v enoti *System.Sensors* (tipala so podprta tudi na namiznih računalnikih!), delo z njimi pa prikazuje primer *SensorInfo*.

Dotik

Zasloni na dotik so podprti s standardnim FireMonkey gradnikom *TGestureManager* (poznamo ga tudi v VCL-ju). Gradnik postavite na obrazec in ga priredite lastnosti *GestureManager* kontrole, ki bo obdelovala kretnje (to je lahko kar sam obrazec). Nato v lastnosti *Gestures* označimo kretnje, ki bi jih radi podpirali, ter jih pripnemo na akcije, ki naj se ob kretnjah izvedejo.



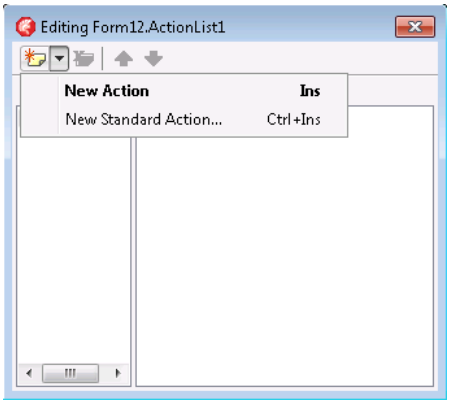
Določimo lahko tudi odziv ob sistemskih kretnjah (povečava z dvema prstoma, vrtenje s tremi ...) v lastnosti *InteractiveGestures*. Nato v dogodku *OnGesture* obdelamo podatke kretnje (*TGestureEventInfo*) in ustrezno reagiramo. Raba interaktivnih kretenj je prikazana v treh primerih v mapi *InteractiveGestures*.

```
procedure TImageRotationForm.FormGesture(Sender: TObject;
  const EventInfo: TGestureEventInfo; var Handled: Boolean);
var
  LObj: IControl;
  LImage: TImage;
begin
  if EventInfo.GestureID = igiRotate then
  begin
    LObj := Self.ObjectAtPoint(ClientToScreen(EventInfo.Location));
    if LObj is TImage then
    begin
      { rotate the image }
      LImage := TImage(LObj.GetObject);
      if (TInteractiveGestureFlag.gfBegin in EventInfo.Flags) then
        FLastAngle := LImage.RotationAngle
      else if EventInfo.Angle <> 0 then
        LImage.RotationAngle := FLastAngle - (EventInfo.Angle * 180) / Pi;
    end;
  end;
end;
```

Alternativno lahko reagiramo na dogodek *OnMouseDown*, ki se sproži ob dotiku.

Standardne akcije

Mnogo dela na mobilnih napravah lahko postorimo tako, da naredimo neko od »standardnih« akcij, ji nastavimo nekaj parametrov in jo izvedemo, tako da jo pripnemo na element grafičnega vmesnika, ali pa pokličemo njeno lastnost *ExecuteTarget*.



Standardno akcijo najlažje naredimo tako, da na obrazec položimo gradnik *TActionList*, ga dvokliknemo in z zgornjo levo ikono izberemo *New standard action*. Pojavi se obširen seznam akcij, iz katerega bomo izpostavili nekaj najbolj zanimivih.

TVirtualKeyboard na zaslon prikliče virtualno tipkovnico.

TTakePhotoFromLibrary prikaže privzeti program za pregledovanje slik, iz katerega si lahko uporabnik izbere sliko. Kontrola se nato prenese nazaj v naš program, kjer se sproži dogodek akcije *OnDidFinishTaking*.

TTakePhotoFromCameraAction prikaže privzeti program za fotografiranje, kjer lahko uporabnik naredi nov posnetek. Kontrola se nato prenese nazaj v naš program, kjer se sproži dogodek akcije *OnDidFinishTaking*.

TShowShareSheetAction prikaže seznam programov, ki jim lahko pošljemo podatke. Te moramo predhodno pripraviti v dogodku akcije *OnBeforeExecute*.

TPhoneCallAction sproži aplikacijo za telefoniranje in po želji sproži klicanje številke (nastavimo jo v lastnosti *TelephoneNumber*).

TChangeTabAction zamenja aktivno stran v gradniku *TTabControl* (aktivno stran moramo nastaviti v lastnosti *Tab*). Za premikanje lahko uporabimo tudi animacijo (*Transition*).

Če bi radi to akcijo uporabili za premikanje na poljubno stran (denimo stran desno ali levo od aktivne strani, moramo lastnost *Tabs* nastaviti v kodi).

```
procedure TForm13.SetActionTab;
begin
  if TabControl1.ActiveTab.Index > 0 then
    ChangeTabActionRight.Tab := TabControl1.Tabs[TabControl1.ActiveTab.Index-1];
  if TabControl1.ActiveTab.Index < (TabControl1.TabCount - 1) then
    ChangeTabActionLeft.Tab := TabControl1.Tabs[TabControl1.ActiveTab.Index+1];
end;

procedure TForm13.TabControl1Change(Sender: TObject);
begin
  SetActionTab;
end;
```

Lokalizacija

Možnost lokalizacije programov še ni vgrajena v platformo FireMonkey. V *Virih* (razdelek *Lokalizacija*) smo zbrali seznam dodatkov za Delphi, ki delujejo tudi v FireMonkeyu in na mobilnih platformah.

Storitve platforme

V enoti *FMX.Platform* leži razred *TPlatformServices*, s katerim lahko pridemo do posebnih storitev, ki jih nekatere naprave podpirajo, druge pa morda tudi ne. V primeru *KeyboardToolbar* vidimo, kako koda dostopa do storitve za nadzor virtualne tipkovnice, ki ji lahko s to storitvijo vključimo dodatno orodjarno in na njo postavimo gumb.

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  if TPlatformServices.Current.SupportsPlatformService(
       IFMXVirtualKeyboardToolbarService, IInterface(FService)) then
  begin
    swToolbar.IsChecked := FService.IsToolbarEnabled;
    swDoneButton.IsChecked := FService.IsHideKeyboardButtonVisible;
  end
  else begin
    FService := nil;
    swToolbar.Enabled := False;
    lbButtons.Enabled := False;
    swDoneButton.Enabled := False;
    ShowMessage('Virtual keyboard service is not supported');
  end;
end;
```

Vse storitve so naštete v enoti *FMX.Platform.Android* v metodi *RegisterCorePlatformServices*. Najbolj zanimive med njimi so:

IFMXApplicationEventService – s to storitvijo lahko aplikacija izve, kdaj je postala aktivna ali neaktivna, lahko pa je tudi opozorjena, da bo zaključena ali da sistemu zmanjkuje pomnilnika.

IFMXSystemInformationService, IFMXScreenService, IFMXDeviceService – s temi storitvami pridobimo nekaj pomembnih informacij o sistemu.

IFMXVirtualKeyboardService – omogoča nadzor nad virtualno tipkovnico.

IFMXLoggingService – omogoča zapisovanje v sistemski dnevnik (log).

Tiskanje

Tiskanje iz Androida še ni dobro rešeno. Edino standardno rešitev prinaša Googlova storitev *CloudPrint*. Z njo lahko tiskamo preko omrežja neposredno na tiskalnike, ki podpirajo CloudPrint ali HP-jev ePrint, pa tudi na tiskalnike, za katere v Chromu vzpostavimo poseben strežnik. Žal pa velja, da morata biti tako telefon kot Chrome, ki streže tiskalniku, prijavljena v isti Google račun, kar naredi storitev bistveno manj uporabno, kot bi lahko bila.

V tiskalno storitev *CloudPrint* lahko podatke pošljete s standardno akcijo *TShowShareSheetAction* (glej poglavje *Deljenje podatkov*).

Nekaj povezav na to temo smo zbrali v *Virih*.

Logiranje

Čeprav je v RAD Studio vgrajen razhroščevalnik, ki deluje z Androidnimi aplikacijami, je včasih najboljši način reševanja problemov zapisovanje stanja v dnevnik (log).

V aplikaciji uporabimo enoto *AndroidAPI.Log* ter funkcije *LOGI, LOGW, LOGE, LOGF* (razlikujejo se po nivoju napake – *Information, Warning, Error, Fail*).

```
function LOGI(Text: MarshaledAString): Integer;
function LOGW(Text: MarshaledAString): Integer;
function LOGE(Text: MarshaledAString): Integer;
function LOGF(Text: MarshaledAString): Integer;
```

To so nizkonivojske funkcije, ki podatke pošiljajo neposredno v operacijski sistem. Ta ne pozna koncepta Delphijevih nizov, zato moramo vse podatke tipa *string* pretvoriti v *MarshaledAString*, ki je le kazalec na niz osembitnih znakov. To delo opravi zapis *TMarshaller,* ki je definiran v enoti *System.SysUtils.*

```
var
  LMarshaller: TMarshaller;

LOGI(LMarshaller.AsAnsi('Tukaj smo!').ToPointer);
```

Višjenivojski pristop omogoča storitev *IFMXLoggingService*.

```
uses FMX.Platform;

var
  LoggingService: IFMXLoggingService;
begin
  LoggingService :=
    FMX.Platform.TPlatformServices.Current.GetPlatformService(IFMXLoggingService)
    as IFMXLoggingService;

  if Assigned(LoggingService) then
      LoggingService.Log('Zivijo!', []);
end;
```

Še višji nivo (združljiv z namiznimi aplikacijami) prinaša knjižnica DX Library, ki tudi sicer vsebuje nekaj koristnih dodatkov za mobilno programiranje.

Dnevnik si lahko ogledate v ukazni vrstici z ukazom *adb logcat* ali s programom *Android Log Viewer*.

JavaBridge/JNI

Delphi XE5 omogoča klicanje vseh funkcij operacijskega sistem Android, tudi če niso že vnaprej ovite v pascalsko kodo. Prav tako omogoča povezovanje z javanskimi knjižnicami tretjih avtorjev. Opis postopka presega okvir te skripte, zato bomo na tem mestu predstavili le nekaj osnovnih idej. Več o tej temi lahko izveste v povezavah na koncu skripte.

Za povezovanje je uporabljena javanska tehnologija JNI (Java Native Interface). Vmesnike najdete v enoti *Androidapi.JNI* in podenotah (denimo *Androidapi.JNI.os*).

Ko hočemo oviti javanski razred, moramo pripraviti dva vmesnika in en razred. Oglejmo si primer iz *Androidapi.JNI.os*.

```
JVibratorClass = interface(JObjectClass)
['{00EEB92F-8145-441D-81C3-218E7B271F1B}']
end;

[JavaSignature('android/os/Vibrator')]
JVibrator = interface(JObject)
['{82BDC8BC-22A3-4EAD-99AF-5DA70739B086}']
  {Methods}
  procedure cancel; cdecl;
  function hasVibrator: Boolean; cdecl;
  procedure vibrate(milliseconds: Int64); cdecl; overload;
  procedure vibrate(pattern: TJavaArray<Int64>; repeat_: Integer); cdecl;
    overload;
end;

TJVibrator = class(TJavaGenericImport<JVibratorClass, JVibrator>) end;
```

Prvi vmesnik (*JVibratorClass*) opisuje razredne metode javanskega razreda. V tem primeru je prazen, ker ne obstajajo.

Drugi vmesnik (*JVibrator*) opisuje navadne metode razreda. Vidimo, da moramo vedno uporabiti način prenosa parametrov *cdecl*. Če si ogledamo dokumentacijo operacijskega sistema, vidimo, da so v tem vmesniku le originalne deklaracije, predelane v pascal.

Public Methods	
abstract void	cancel () Turn the vibrator off.
abstract boolean	hasVibrator () Check whether the hardware has a vibrator.
abstract void	vibrate (long[] pattern, int repeat) Vibrate with a given pattern.
abstract void	vibrate (long milliseconds) Vibrate constantly for the specified period of time.

Razred (*TJVibrator*) pripravi vse potrebno za povezavo Delphija z javansko kodo.

V programu moramo najprej narediti sistemski objekt s klicom *SharedActivity.getSystemService*. Nato ta objekt »ovijemo« v pascalski objekt s klicem *TJVibrator.Wrap*. Na tem objektu lahko uporabimo funkcije, deklarirane v razredu *JVibrator*.

```
uses
  Androidapi.JniBridge,
  Androidapi.Jni.os,
  Androidapi.Jni.JavaTypes,
  Androidapi.Jni.App,
  FMX.Helpers.Android;

var
  VibratorObj: JObject;
  Vibrator: JVibrator;

begin
  VibratorObj := SharedActivity.getSystemService(
    TJActivity.JavaClass.VIBRATOR_SERVICE);
  Vibrator := TJVibrator.Wrap((VibratorObj as ILocalObject).GetObjectID);

  Vibrator.vibrate(1000);
end;
```

Klicanje drugih aplikacij

Z nizkonivojskimi metodami lahko iz svoje aplikacije kličemo druge in jim prepustimo delo. Na tak način denimo odpremo povezavo URL v brskalniku. V Androidu se takemu početju reče *intent*, vmesnik *JIntent*, ki to omogoča, pa je deklariran v *Androidapi.JNI.GraphicsContentViewText*.

```
uses
  Androidapi.JNIBridge,
  Androidapi.JNI.JavaTypes,
  Androidapi.JNI.GraphicsContentViewText,
  Androidapi.JNI.Net,
  FMX.Helpers.Android;

procedure LaunchURL(const URL: string);
var
  Intent: JIntent;
begin
  Intent := TJIntent.JavaClass.init(TJIntent.JavaClass.ACTION_VIEW,
    TJNet_Uri.JavaClass.parse(StringToJString(URL)));

  SharedActivity.startActivity(Intent);
end;
```

S to funkcijo lahko tudi pokličete telefonsko številko (namesto prefiksa *http://* uporabit *tel://*), pošljete SMS (*sms://*) ali elektronsko poštno sporočilo (*mailto://*), tvitnete (*twitter://*), odprete zemljevid na določenih koordinatah (*geo://*) in še kaj.

Baze podatkov

Do podatkovnih baz lahko na mobilni napravi dostopamo na dva načina. Podatkovne baze so lahko shranjene na napravi in uporabljene za shranjevanje nastavitev in podatkov ter za delo brez povezave. Lahko pa se povežemo tudi z oddaljenimi bazami, kar naredimo z razdelitvijo aplikacije na dva ali tri sloje. V obeh primerih uporabljamo za prikaz podatkov ogrodje Visual LiveBindings, za dostop do podatkov pa gonilnike FireDAC in tehnologijo DataSnap, o katerih smo že imeli nekaj predavanj (v Virih najdete povezave na skripte in diapozitive iz teh predavanj).

Mobilne baze podatkov

Delphi podpira dve mobilni bazi podatkov – SQLite in InterBase. Prvi je že vgrajen v operacijski sistem Android (torej ne zahteva nobene dodatne namestitve) in je zastonj, je pa relativno omejen in primeren le za enostavno shranjevanje podatkov. InterBase dobite v dveh različicah – omejeni (a zastonj) IBLite in polni (plačljivi) InterBase ToGo. Oba imata implementiran pravi strežnik SQL, ki podpira standard SQL-92. Za dostop do podatkov vedno uporabljamo gonilnike FireDAC.

Primerjavo vseh treh rešitev prikazuje spodnja tabela.

SQLite	IBLite	InterBase ToGo
Zastonj	Zastonj	Komercialen
Omejena funkcionalnost	Omejena funcionalnost	Polna funkcionalnost
Brez šifriranja	Brez šifriranja	Šifriranje podatkov
Enostavna shramba podatkov	Polna podpora SQL-92	Polna podpora SQL-92
Zaporedno branje/pisanje	Hitro sočasno branje/pisanje	Hitro sočasno branje/pisanje

Dvo- ali večslojne aplikacije

Večslojne aplikacije lahko sprogramiramo z gonilniki FireDAC, ki dostopajo neposredno do podatkovnega strežnika, ali pa z gonilniki FireDAC, ki dostopajo do strežnika DataSnap (napišemo ga v Delphiju in teče na namiznem računalniku z operacijskim sistemom Windows), ta pa do podatkovnega strežnika.

REST

Aplikacije, ki dostopajo do oddaljenega vira podatkov, lahko za dostop uporabijo tudi novo odjemalsko knjižnico REST. Ta omogoča:

- Asinhrono izvajanje klicev
- Rabo posredniških (proxy) strežnikov
- Varne povezave (https)
- Overovljanje (basic, OAuth1, OAuth2)
- Pretvorbo JSON podatkov v TObject in nazaj
    - JSON to TObject, TObject to JSON
- Povezovanje z Visual LiveBindings

Viri

Namestitev

SysCheck

http://play.google.com/store/apps/details?id=com.ss.syscheck

Namestitev

http://docwiki.embarcadero.com/RADStudio/XE5/en/Android_Mobile_Application_Development

Priprava emulatorja

http://docwiki.embarcadero.com/RADStudio/XE5/en/Running_Your_Android_Application_on_an_Android_Emulator

»Delphi XE5 Android Up and Running«

http://www.youtube.com/watch?v=OD4StmnUEUA

Gonilniki

Seznam gonilnikov po proizvajalcih

http://developer.android.com/tools/extras/oem-usb.html
http://www.clockworkmod.com/carbon/drivers

Univerzalni gonilniki za Google, Samsung, Asus in HTC

http://forum.xda-developers.com/showthread.php?t=2263822

Zrcaljenje in oddaljeni dostop

VMLite Android App Controller

http://www.vmlite.com/vaac/

Droid VNC server

http://play.google.com/store/apps/details?id=org.onaips.vnc

Droid@Screen

http://droid-at-screen.ribomation.com

Android Screen View

http://delphi.org/2013/09/android-screen-view/

Programiranje

Vaje

http://docwiki.embarcadero.com/RADStudio/XE5/en/Mobile_Tutorials:_Delphi_Mobile_Application_Development_(iOS_and_Android)

Primeri (izvorna koda)

http://sourceforge.net/projects/radstudiodemos/

Priročnik za izdelavo nove mobilne aplikacije

http://docwiki.embarcadero.com/RADStudio/XE5/en/Creating_an_Android_App

Slike z več ločljivostmi

http://docwiki.embarcadero.com/RADStudio/XE5/en/Using_Multi-Resolution_Bitmaps

Migracija namiznih programov

http://docwiki.embarcadero.com/RADStudio/XE5/en/Migrating_Delphi_Code_to_Mobile_from_Desktop

Knjižnice in triki

Dostop do odložišča

http://delphihaven.wordpress.com/2013/07/22/fmx-tclipboard-and-tmacpreferencesinifile-xe4/

Nastavitve programa v obliki INI datotek

http://delphihaven.wordpress.com/2013/09/12/a-few-xe5-related-bits/

Preverjanje, ali obstaja internetna povezava

http://delphi.radsoft.com.au/2013/11/checking-for-an-internet-connection-on-mobile-devices-with-delphi-xe5/

DX Library

http://www.monien.net/dx-library-utility-functions-for-ios-and-other-platforms-available-for-free-via-subversion/

Logiranje in razhročevanje

Android Debug Monitor

http://developer.android.com/tools/debugging/ddms.html

adb

http://developer.android.com/tools/help/adb.html

adb logcat

http://developer.android.com/tools/help/logcat.html

Android Log Viewer

http://bitbucket.org/mlopatkin/android-log-viewer/

Namestitev iz ukazne vrstice

http://delphi.org/2013/11/installing-and-running-android-apps-from-command-line/

Lokalizacija

Multilizer

http://www2.multilizer.com/

Sisulizer

http://www.sisulizer.com/

TsiLang Components Suite

http://www.tsilang.com/

delphi-package i18n

http://www.delphiarea.com/products/delphi-packages/i18n/

GNUGetText for Delphi

http://dxgettext.po.dk

Tiskanje

CloudPrint

http://www.google.com/cloudprint
http://lifehacker.com/5775462/
http://lifehacker.com/5742035/
http://gmailblog.blogspot.com/2011/01/print-from-your-phone-with-gmail-for.html

JavaBridge/JNI

Kako odpreti URL v drugi aplikaciji

http://delphi.org/2013/10/sending-a-url-to-another-app-on-android-and-ios-with-delphi-xe5/

Povezovanje z zunanjimi javanskimi knjižnicami

http://plus.google.com/+PaulFoster/posts/Tc7viMithGi
http://www.pclviewer.com/android/androidJNI.html

Android 2 Delphi

http://chuacw.ath.cx/blogs/chuacw/archive/2013/10/24/the-story-of-the-win32-port-of-the-android-jni-framework.aspx

Delphi program kot storitev

http://blog.blong.com/2013/11/delphi-and-android-services.html
http://blog.blong.com/2013/11/delphi-and-android-services-part-2.html

Starejša predavanja

FireMonkey2 - nova različica ogrodja za izdelavo vizualnih uporabniških vmesnikov

Predstavitev: http://17slon.com/EA/EA-FireMonkey2.pps
Skripta: http://17slon.com/EA/EA-FireMonkey2.pdf
Programi: http://17slon.com/EA/EA-FireMonkey2.zip

Embarcadero RAD Studio in baze podatkov (DataSnap & FireDAC)

Predstavitev: http://17slon.com/EA/EA-DataSnap_FireDAC.pps
Skripta: http://17slon.com/EA/EA-DataSnap_FireDAC.pdf
Programi: http://17slon.com/EA/EA-DataSnap_FireDAC.zip

Visual LiveBindings - ogrodje za povezovanje z bazami podatkov

Predstavitev: http://17slon.com/EA/EA-LiveBindings.pps
Skripta: http://17slon.com/EA/EA-LiveBindings.pdf
Programi: http://17slon.com/EA/EA-LiveBindings.zip

Video

CodeRage 8

http://www.embarcadero.com/coderage/embtc-gated-sessions1013

Accessing the Android API

http://www.youtube.com/watch?v=GcuYc7F0lIU

Beyond the App

http://www.youtube.com/watch?v=msoKStc-tRM

Cross Platform Secure Database Storage for Mobile & Desktop

http://www.youtube.com/watch?v=pYj_WTxiv1U

Deep Dive into Creating Android Apps

http://www.youtube.com/watch?v=Iq0LO3gcbUg

Developing a Simple Mobile Game with Firemonkey

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

Devices & Sensors in iOS & Android

http://www.youtube.com/watch?v=O-0DmmtwpEw

Designing Common User Interfaces for iOS & Android

http://www.youtube.com/watch?v=Oypth7HCjZM

Effectively Using List Controls in Mobile Apps

http://www.youtube.com/watch?v=XRj3qjUjBlc

From Idea to Submitted Mobile App in 30 Minutes

http://www.youtube.com/watch?v=gUv5Th_QPfk

Introduction to iOS and Android Devices Connecting to a DataSnap Server

http://www.youtube.com/watch?v=vHthFtUS_I0

Real Mobile Implementation Patterns

http://www.youtube.com/watch?v=QmbV4rAuZL4

The New REST Client Library: A Tool of Many Trades

http://www.youtube.com/watch?v=nPXYLK4JZvM

Responsive Delphi Design, Dive into the Details

http://www.youtube.com/watch?v=97rFMCe4-7I

The New REST Client Library

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

What's New in FireMonkey for XE5

http://www.youtube.com/watch?v=rZbYp3ULnsc