Fehér Krisztián

Androidos szoftverfejlesztés Flexben alapfokon

ISBN 978-615-6184-35-1

Elérhetőség: feher.konyvek@gmail.com

Tartalomjegyzék

Előszó

A mobileszközök térnyerését nem lehet nem észrevenni. Ismerős a kép: utazás közben az emberek szeretik mobileszközeikkel tölteni az időt. A jelenség általános és mára tömeges méretet öltött.

A könyv alapját egy előadás szolgáltatja, melyet a téma iránt mélyebben érdeklődő informatikusoknak tartott a szerző. Az előadás ötletét szintén a fenti megfigyelés adta. Az előadás címe ez volt: *„Androidos fejlesztés modern eszközökkel"*. Az előadás anyagát felhasználva, azt jelentősen kibővítve született meg ez a könyv.

Kiknek szól ez a könyv?

A könyv teljesen kezdő és haladó fejlesztőknek, informatikusoknak ajánlható, akik most ismerkednek a mobilfejlesztés világával. Az alapvető informatikai ismeretekre ugyan szükség van, a mélyebb programozási ismeretek birtokában pedig gyorsabban haladhatunk, de ez nem feltétele a könyv ismeretanyagának a sikeres feldolgozásához, programozáshoz.

Miben történik a fejlesztés?

A könyvben az **Adobe AIR**- és **Apache Flex**alapú mobilprogramozásról olvashatunk, melynek alapnyelve az **ActionScript**. A könyv bemutatja az **Adobe Flash Builder** fejlesztőeszközt és egy lehetséges megközelítési módját annak, hogy az AIR alkalmazások hogyan képesek kihasználni a **PHP** és a **MySQL** lehetőségeit.

Hazánkban alapvetően a Java-alapú fejlesztéseket részesítik előnyben, így a mobilfejlesztés és az AIR kapcsolatáról méltánytalanul szinte semmilyen magyar nyelvű forrást nem találunk, noha az AIR segítségével kiváló mobilalkalmazásokat készíthetünk Androidra, vagy akár iOSre is.

Az **ActionScript** alapon végzett fejlesztés sok előnnyel jár. Gyorsan tanulható, lényegesen egyszerűbb a Java-alapú fejlesztésnél és univerzálisabb tudást szerzünk, hiszen az ActionScript nem csupán mobilfejlesztésre használható, hanem webes, vagy asztali alkalmazásokat is ugyanolyan könnyedén készíthetünk vele (az ugyanolyan itt szó szerint értendő). A készíthető alkalmazások minősége pedig semmiben sem marad el a Java-ban fejlesztett alkalmazásokétól.

A könyv felépítése

A könyv négy tematikus részre osztható: egy elméleti, egy az ActionScriptet és a Flex-et, és egy, a példaalkalmazások készítését bemutató gyakorlati blokkra, valamint zárásként egy kódrecept gyűjteményre.

Az első, elméleti rész azokat a szükséges ismereteket tartalmazza, melyekre minimálisan szükség van az induláshoz és segítségükkel otthonosan mozoghatunk a témában.

A második részben az ActionScript és a Flex olyan szintű bemutatására törekszünk, melyekkel a könyvben tárgyalt példaprogramok elkészítése tökéletesen érthető, egyben biztos kiindulási alapot teremt a további fejlődéshez. Ezt hivatott segíteni a könyv végi irodalomjegyzék is, melyben csupa hasznos, valóban használható könyv listája található. A könyv példái a PHP szkripteket és MySQL lekérdezéseket is használnak, ám a PHP és MySQL külön ismertetésétől – annak szerteágazó volta miatt – eltekintünk, cserébe viszont bőséges magyarázatokkal látunk el minden szkriptet. A könyv végén található irodalomjegyzék tartalmaz több kiváló könyvet is, melyekkel elmélyedhetünk a PHP és a MySQL világában.

A harmadik, gyakorlati rész példaalkalmazások elkészítését mutatja be, lépésről-lépésre. Először csak egyszerű, de azonnal futtatható alkalmazásokat készítünk, majd a könyv végére pedig egy PHP-n keresztül MySQL adatbázissal kommunikálni képes alkalmazást is készíteni fogunk. Egy adatbázissal kommunikáló mobilalkalmazás megírásának képessége komoly és hasznos alkalmazások elkészítéséhez teremt megfelelő alapot, ezért lett a könyv célja ez. A fejezetekben számos tippet és trükköt is talál a Tisztelt Olvasó a PHP és a MySQL használatához, melyeknek mobilfejlesztésnél nagy hasznát vehetjük.

A negyedik, záró részben a megszerzett tudás további elmélyítésére inspiráló ”instant” kódpéldákat találunk, melyekkel még sokoldalúbbá tehetjük alkalmazásainkat.

A könyvben használt jelölések

A könyvben feliratok, elnevezések aposztrófok közé téve jelennek meg (például: 'felirat'). A forráskódok, programozási nyelvek elemei más betűtípussal vannak szedve a jobb olvashatóság érdekében (például: forráskód).

Ahol lehetséges, a Flash Builder szintaktikai szövegkiemeléseit megtartottuk a könnyebb olvashatóság érdekében, esetenként a fontosabb kódrészleteket külön vastagon szedve kiemelten.

A könyvben közölt forráskódok sorait azok szélessége miatt tördelni kellett, esetenként erre ügyelni kell a kódok értelmezésekor, ill. begépelésekor.

A könyv legvégén ajánlott könyvek és webhelyek bőséges listája található, melyekkel tovább bővíthetjük megszerzett ismereteinket.

Fedezzük hát fel az androidos fejlesztés világát!

1. Felkészülés a fejlesztésre

1.1. Az Adobe AIR platformról

Az AIR az **Adobe Integrated Runtime** elnevezés rövidítése és egy olyan technológiát takar, mellyel Flash-alapú alkalmazásokat böngészőn kívül, az operációs rendszer natív alkalmazásaként lehet futtatni. Ezzel a Flash rugalmasságát a natív alkalmazások előnyeivel (pl. hozzáférés a lokális erőforrásokhoz) lehet ötvözni.

A programozás során használt programozási nyelv, az **ActionScript**, ECMA szabványra épülő nyelv, akárcsak a Java. Az Actionscript lehetővé teszi, hogy a Flash példátlanul látványos multimédiás képességeit interaktivitással ruházzuk fel.

Az AIR alkalmazások készítését külön SDKval lehet elvégezni. Az AIR SDK ingyenesen letölthető az Adobe weboldaláról:

(**http://www.adobe.com/devnet/air/air-sdk-download.html**), ám leghatékonyabban valamilyen integrált fejlesztői környezettel lehet kihasználni a technológia összes képességét.

Az AIR fejlesztések páratlan előnye, hogy ugyanazzal a kódbázissal többféle platformra készíthetünk alkalmazásokat, a kód újraírása nélkül. Így, aki megtanul AIR alkalmazásokat írni, „egy csapásra" képes Windows, Mac OSX, Android, Blackberry, iOS, de még SmartTV rendszerekre is alkalmazásokat készíteni.

Az ActionScript-alapú fejlesztésekkel karöltve zajlott a Flex keretrendszer fejlesztése is. A Flex nem más, mint egy ActionScriptben megírt osztálygyűjtemény és architektúra, mely elsősorban az alkalmazások kezelőfelületének az elkészítésére használható. A Flex forráskódja XML alapú és nagyon könnyen olvasható. Az Adobe a Flex keretrendszer gondozását 2012ben az Apache alapítványnak adta át.

Az ActionScript aktuális verziója ezen könyv megírásakor a 3.0-ás, a Flex-é 4.11.

Mobilalkalmazásokat Flex és tisztán ActionScript alapon, vagy keverten is fejleszthetünk. Ez utóbbi megoldást mutatja be a könyv is.

1.2. Az Androidról és a statisztikákról

Naponta átlagosan 1 millió új, Android-ot futtató mobileszközt aktiválnak.

Az Android egy UNIX alapokon nyugvó operációs rendszer, lényegében a Linux egy változata. Natív programozási nyelve a Java. Az Android Inc.-t 2003-ban alapították, 2005ben ezt vette meg a Google és az ún. Open Handset Alliance, mely számos vállalatot, köztük sok telekommunikációs céget foglal magában.

Az Android első kiadása 2007-ben jelent meg, de kereskedelmi forgalomban kapható mobileszközön 2008-ban volt először kapható.

Többféle verziójú Android található a forgalomban levő mobileszközökön, a legelterjedtebbek a 2.3, 4.0.x és 4.1.x verziók, durván egymáshoz hasonló arányban. Az arányok természetesen fokozatosan tolódnak az újabb verziók felé.

A legkedveltebbek a mobiltelefonok és a 7,58 cm képátmérőjű, nagy pixelsűrűségű kijelzőkkel rendelkező eszközök.

Miért fontosak ezek az információk? A következő fejezetből megtudjuk.

1.3. A mobilfejlesztés fő szempontjai

1.3.1. A mobilalkalmazásoknál a minőség kiemelt szempont

Az asztali számítógépek világban már szinte megszoktuk, hogy a szoftverek időről-időre „kifagynak" és a legkülönfélébb hibákat produkálják. Egy mobilalkalmazás esetében ez nem megengedhető. A felhasználók a hibás mobilalkalmazásoktól rövid úton megszabadulnak és ezt joggal teszik: a kínálat bőséges, van miből válogatni.

1.3.2. A mobilvilág más

Ahogyan egy desktop alkalmazás és egy webes alkalmazás közé sem tehetünk egyenlőségjelet, ugyanúgy egy mobilalkalmazás elkészítésekor sem járhatunk el egy az egyben úgy, mint egy webalkalmazásénál (pl. erőforrásokkal történő gazdálkodás).

1.3.3. Tudatosan kezeljük a jogosultságokat!

Az Android szigorúan szabályozott keretek között futtatja az alkalmazásokat. Ez több szinten is tetten érhető.

Alacsony szinten minden alkalmazáshoz egy virtuális gép rendelődik (a Dalvik), az Android virtuális gépe. Az alkalmazások nem kommunikálhatnak egymással közvetlenül, hanem az operációs rendszer közvetítésével, üzenetek segítségével. Számos más tényező is a biztonságot hivatott növelni.

Az alkalmazások szintjén pedig az alkalmazásnak előre tudatnia kell az operációs rendszerrel, hogy a készülék mely funkcióihoz kíván hozzáférni, tipikusan ilyen például az internethozzáférés, a telefonhívások, a beépített kamera használata stb.

Az ökölszabály az, hogy csak olyan jogosultságokat igényeljen az alkalmazásunk, melyekre feltétlenül szüksége is van!

A túlzott és felesleges jogosultságigénylés hátulütője az, hogy a felhasználók annál bizalmatlanabbak egy alkalmazással szemben, minél több jogosultságot igényel az. (Ha egy sima játék hozzáférést akar például a telefonhívásokhoz, úgy az óvatosabb felhasználó gyanút fog, és nem telepíti azt.)

1.3.4. Ésszerű keretek között tartott kompatibilitás

Napjainkban egyfajta közmegegyezés van arról, hogy az Android 2.2/2.3-as verzióival érdemes fenntartani a kompatibilitást. Java-alapú fejlesztések esetében ez inkább a 2.2-es verziót szokta jelenteni.

Adobe AIR esetében a 2.3-as verzióig viszszamenően kompatibilis alkalmazásokat készíthetünk.

Érdemes abból kiindulni, hogy milyen ésszerű ráfordítás szükséges a kompatibilitás megőrzéséhez. Nincs ez másként az asztali számítógépek világában sem. Ha túl alacsonyra tesszük a mércét, az felesleges költségeket és ráfordítást generál. Ha túl magasra, akkor potenciális felhasználóktól eshetünk el.

1.3.5. Takarékoskodni az erőforrásokkal

A mobileszközök hardverei szédületes tempóban fejlődnek. Ma már a többmagos CPU-k és a nagy teljesítményű grafikus gyorsítók is átlagosnak számítanak. Érdemes megjegyezni, hogy ez a fejlődés mennyire hasonlóan zajlik a desktop világban végbemenő fejlődéshez és hogy a mobilvilágban ez a fejlődés sokkal rövidebb ciklusokkal megy végbe.

Talán hamarosan eljön a nap, amikor mindegy lesz, hogy egy okostelefont, vagy egy desktop PC-t fogunk használni internetezésre, vagy akár munkára is. A tejjel és mézzel folyó Kánaánra, amiben nem kell törődnünk a mobileszközök erőforrásainak kímélésével, azonban még várnunk kell és a jelenben igenis fontos, hogy a többnyire még mindig szerényebb teljesítményű eszközökön is közel hasonló felhasználói élményt nyújtson alkalmazásunk, mint egy erősebb hardverrel rendelkező eszközön.

1.3.6. A v7-es

Az androidos eszközök ún. ARM processzorokat használnak, amelyek RISC processzorok. Liszensz alapján több gyártó is készít ilyen CPU-kat. Ezek utasításszinten 100 százalékig kompatibilisek egymással. **Az Adobe AIR alkalmazások a v7-es utasításkészletű CPU-kkal kompatibilisek.** A v7-es CPU-k már jó ideje bőséges számban jelen vannak a piacon és az új eszközökben is ilyeneket találunk. Mindazonáltal rengeteg olyan – elsősorban régebben gyártott – készülék van forgalomban, amelyek pl. v5-ös utasításkészletű CPU-val rendelkeznek. Erre oda kell figyelni.

1.3.7. A 3D-s megjelenítés mobileszközökön is lehetséges

A CPU-k fejlődése mellett a GPU-k is rohamosan erősödnek a mobileszközökben. Ezek a grafikus gyorsítók fejlett OpenGL támogatással is rendelkezhetnek. Az Adobe AIR alkalmazások teljes mértékben ki tudják aknázni a GPUkban rejlő lehetőségeket.

1.3.8. Ez a RAM nem az a RAM

A hirdetésekben gyakran találkozhatunk a mobileszközök „RAM" specifikációival, melyet hallva önkéntelenül is a desktop világban használatos memóriamodulokra gondolunk.

Nos, a helyzet ennél bonyolultabb. A kétféle RAM elnevezés ugyanis más-más memóriatípust takar. A mobileszközök memóriája ugyanis Flash memória, amely megőrzi tartalmát áramellátás hiányában is. Hasonló memóriákat használunk például digitális fényképezőgépekhez is.

Az androidos eszközök két memóriát használnak. Egy ún. belső tárolót és egy külső tárolót (mindkettő az eszközbe beépített memóriát takar). Utóbbi mérete általában többszöröse a belső tárolóénak. A dolgokat tovább bonyolítja, hogy a belső tároló különböző logikai egységekre, ún. szegmensekre van osztva. A belső memória szegmensei hasonlatosak a desktop PCkben használt merevlemezek partícióihoz. Ráadásul a belső tároló is rendelkezhet külső tárolónak nevezett szegmenssel. Ezeket a mobileszköz operációs rendszere és a telepített alkalmazások használják és csak korlátozott mértékben, vagy egyáltalán nem hozzáférhetőek a felhasználói programok számára.

1.3.9. Itt repül a kismadár!

A mobileszközök a legtöbb esetben legalább egy hátlapi kamerával rendelkeznek. Amennyiben előlapi kamera is van, annak felbontása általában lényegesen alacsonyabb a hátlapi kameráénál. A kamerák által produkált képminőség elég nagy szórást mutat, de azért többnyire sikerrel rögzíthetünk pár képet, a kamerák által látott és rögzített képet pedig felhasználhatjuk alkalmazásainkban.

1.3.10. Szenzorok: izgalmas technológiák egy helyen

Érintőképernyő: a legtriviálisabb kezelőszerve az androidos készülékeknek maga a kijelző. Manapság már lehetséges több ujjal végzett érintések értelmezése is. (Valójában azonban a kétujjas érintéseket is inkább ritkábban használják a felhasználók.)

Gyorsulásmérő: segítségével a készülékek képesek megállapítani, hogy függőlegesen, vagy vízszintesen tartjuk-e őket, továbbá a mozgás irányát is képesek kiszámítani.

Irányérzékelő: ritkábban találkozhatunk vele. Segítségével a készülék képes felismerni, hogy melyik égtáj felé tartják.

(A-)GPS: Néhány kivételtől eltekintve, az androidos eszközökben található GPS-vevő nem azonos hardvert takar a hagyományos navigációs eszközökben található hardverhez képest. Az „A" az „assisted" szó rövidítése és olyan technológiát takar, amellyel lényegesen egyszerűbben megoldható a GPS-alapú helymeghatározás azáltal, hogy a telekommunikációs szolgáltatók hálózatukon keresztül biztosítanak olyan kiegészítő funkciókat, melyekkel a saját hely meghatározása és a GPS műholdak követése megbízhatóan lehetséges szerényebb és kisebb fogyasztású GPS-hardverrel is. Ez egyben azt is jelenti, hogy aktív internetkapcsolat szükséges az A-GPS gördülékeny használatához. Ezt leszámítva azonban a helymeghatározás elérhető minősége kiváló, néhány méteres pontosság is könnyen elérhető.

1.3.11. Akkumulátor: Bírni szusszal…

A mobileszközök egyik legvonzóbb tulajdonsága maga a mobilitás. Nem helyhez kötöttek, hanem akkumulátorról is képesek üzemelni. Az akkumulátorok kapacitása viszont véges. Egy rosszul tervezett alkalmazás jóval nagyobb energiafogyasztást generál, mint egy hatékonyabban megírt. A helyzet ismerős lehet a notebook-ok világából.

1.3.12. FullHD-n innen és túl…

A pixelsűrűség (DPI vagy PPI) a kijelzők azon tulajdonsága, mely azt adja meg, hogy a képernyő egy 2,54cm széles képsora hány darab elemi képpontot, azaz pixelt tartalmaz. A 160as DPI teljesen átlagosnak tekinthető, de 600 feletti DPI értékkel rendelkező kijelzők is léteznek. A sokféleség kavalkádjában a mobilalkalmazásoknak (legalábbis a jól megírtaknak) minden kijelzőn közel azonos módon kell tudniuk megjelenni. Nem triviális ugyanis, hogy például egy nyomógomb ugyanolyan méretben jelenik meg különböző tulajdonságokkal rendelkező kijelzőkön.

Noha a FullHD-s kijelzők sem számítanak már ritkaságnak, még mindig rengeteg olyan készülék vásárolható ill. van forgalomban, melyek kijelzője csupán 320x240, vagy hasonló felbontással rendelkezik.

A megfelelő, felbontásfüggetlen képi megjelenítés nagyon komoly kérdés.

Természetesen a képarányok tovább nehezítik a helyzetet. A megoldás fejlesztői szempontból mindig a gondos, előre elvégzett tervezés.

1.3.13. Képfeldolgozás: tárhely kontra sebesség

Az asztali gépek világából ismert dilemma a mobileszközöknél is felmerül. A fejlett GPUknak köszönhetően ugyanis a képmegjelenítéshez alkalmazási területtől függően választhatunk vektoros megjelenítést, ami helytakarékos, de számításigényes, vagy bittérképalapú megjelenítést, ami ugyan kevés számítási kapacitást igényel, ellenben jóval tárhelyigényesebb.

1.3.14. Nagyfokú hibatűrés

Mobileszközöknél bizony könnyen előfordulhat, hogy a hálózati kapcsolat elvész. Ilyen esetekben – természetesen az adott alkalmazástól függően – akár az adatvesztés lehetőségével is szembe kell nézni, pontosabban ennek elkerülésére kell kidolgozni alkalmazásszintű stratégiát.

Az adatvesztés elkerülése mellett az alkalmazásoknak egészen egyszerűen azzal a helyzettel is kell tudniuk kezdeni valamit, hogy elvész a kapcsolat és újra „fel kell tudniuk venni a fonalat”, ha a kapcsolat helyreáll.

1.3.15. A teljesítménytervezés lehetőségei

Gyakran olvashatunk teljesítménytesztekről, melyek a mobileszközök megjelenítési képességeit mérik. Ezek a kétségkívül hasznos mérések azonban közel sem mutatnak teljes képet egy eszköz teljesítményéről. Léteznek más típusú tesztek is, melyek szintén nagyon fontos információkkal szolgálhatnak fejlesztők számára.

Ezek a tesztek a háttértárra történő írás, sztringműveletek sebességét és más hasonlókat mérnek. Ezek egy részét a Flash Builder is tartalmazza, de rengeteg harmadik fél által készített lehetőség is rendelkezésre áll. A függelékben több ilyen teszteszköz elérhetőségét is megtaláljuk.

1.4. Mire lesz szükségünk?

A modern alkalmazások legtöbbje nem csak úgy önmagában működik, hanem nagy menynyiségű lokális és/vagy hálózati erőforrásokon található adatot dolgoz fel.

Tekintsük át, hogy a könyv ismeretanyagának tárgyalása során milyen megoldásokat használunk majd arra, hogy példaalkalmazásaink számára adatokat szolgáltassunk és feldolgozhassuk azokat!

1.5. MySQL

Napjainkban az ún. relációs adatbázisok dominanciája az adatfeldolgozásban vitán felül áll. Az adatszolgáltatás az adatokat feldolgozó alkalmazások szemszögéből úgy működik, hogy egy alkalmazás egy távoli szerveren futó adatbázis-kiszolgálóalkalmazáshoz küld kéréseket, amely azokat végrehajtja és a kérések alapján rendelkezésre bocsájtja a kért adatokat. A kiszolgáló fejlesztői szemmel nézve lehet egy lokálisan futó natív alkalmazás, egy böngészőben futó alkalmazás, vagy éppenséggel egy mobilalkalmazás is. Könyvünk példái során az ingyenes verzióval is rendelkező **MySQL** szervert fogjuk használni Windows környezetben.

A MySQL-t parancssorból is használhatjuk, de különböző (szintén ingyenes) segédprogramokkal rendkívül könnyedén elvégezhetünk komplex adminisztrációs és konfigurációs feladatokat.

1.6. PHP és Apache

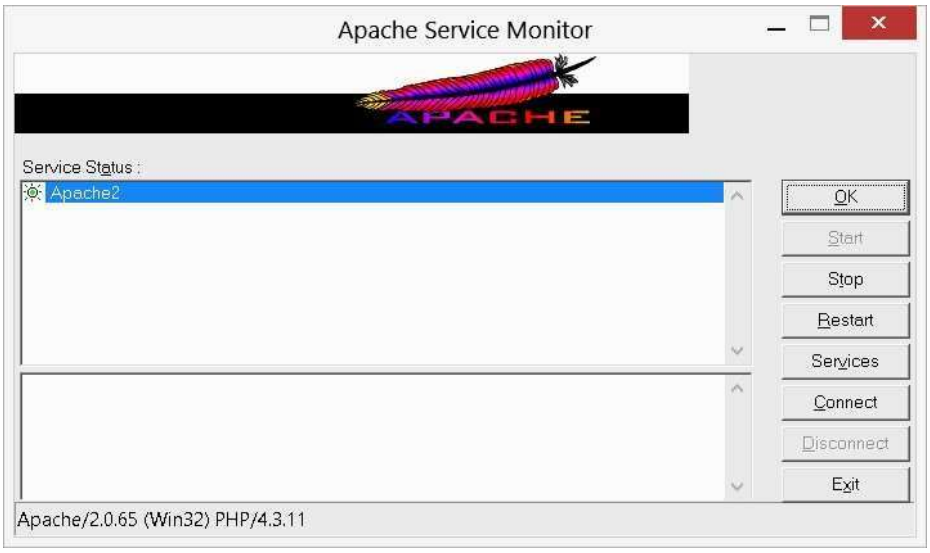
Az adatok tárolásának a módja tehát már megvan. De hogyan férünk hozzá az adatokhoz?

Távoli adatbázisban tárolt adatokhoz mobileszközről közvetlenül nem tudunk hozzáférni, ezért egy közvetítő rétegre lesz szükségünk, mely kommunikálni tud az adatbázissal és az alkalmazásunkkal is. Erre a **PHP-t**, ill. a PHP szkriptnyelvet fogjuk használni.

A MySQL szerverhez hasonlóan a PHP-t is a kiszolgálói oldalon kell telepíteni és beállítani. A kéréseket a szerveren elhelyezett szkriptek hívásával bonyolítjuk le és szintén ezek a szkriptek szolgáltatják majd mobilalkalmazásunk számára az adatokat. A kétirányú kommunikáció képessége nagyon fontos: tudnunk kell adatokat küldeni és fogadni is. Ahhoz azonban, hogy a szerver a PHP-

szkripteket fel is tudja dolgozni, szükség lesz egy ún. http-, vagy webszerverre, ami lényegében egy távoli gépen futó kiszolgálóprogram és a PHP szkriptek futtatását képes elvégezni alkalmazásunk számára. Az egyik legjobb megoldás erre a szintén ingyenes **Apache** kiszolgáló.

Az alábbi képen a 4.3.11-es PHP-t kezelő online 2.0.65-ös verziójú Apache szerver monitorozó ablaka látható.



A MySQL, PHP és Apache telepítésére és megfelelő konfigurálására könyvünk nem tér ki, a könyv végén található irodalomjegyzékben azonban több, a konfigurálást részletesen is bemutató hazai könyv adatait is megtaláljuk.

Néhány jótanácsot azonban mindenképpen fontosnak tartunk megemlíteni.

1.6.1. Verziódzsungel

Mindhárom fent említett szoftverből többféle verzió is létezik, ám nem mindegyik képes együttműködni a másik kettővel. Ezért konzekvensen korábbi, vagy újabb verziókat érdemes használni. Felmerülhet a kérdés: „Hogyan jöhet egyáltalán szóba korábbi verzió?” A válasz attól függ, hogy az adatbázisszervert mi magunk akarjuk-e üzemeltetni, vagy ezt külső szolgáltatóra bízzuk.

Amennyiben a fejlesztés alatt és az üzemeltetés során is mi biztosítjuk a szervert, érdemes a legújabb verziókat használni.

Külső szolgáltató kész megoldásának igénybevételekor már más a helyzet. Több hazai tárhelyszolgáltató ugyanis például a PHP 4.x verzióját használja és a MySQL szerverből sem a legújabbat. Amennyiben ez utóbbi megoldás mellett döntünk, akkor a fejlesztés során figyelembe kell vennünk a szolgáltató oldalán üzemeltetett verziókat.

Külső szolgáltatónál arra is van lehetőség, hogy a szerver felépítését mi vegyük át.

Nagyon fontos: egyes tárhelyszolgáltatók korlátozhatják, szabályokhoz köthetik a szervereiken tárolt adatok, szkriptek stb. hozzáférésének, végrehajtásának lehetőségeit. Külső szolgáltató szolgáltatásainak igénybevétele előtt ezért mindig tájékozódjunk a lehetőségekről!

Ökölszabályként leszögezhetjük: lehetőségeink és igényeink függvényében mindig a lehető legújabb verziókat használjuk!

Könyvünk példáinál a **MySQL 5.2-es,** az **Apache 2.0.65-ös** és a **PHP 4.3.11-es**, verzióit fogjuk használni. Ezek a verziók egyfajta arany középútként jól lefedik a hazai ingyenes tárhelyszolgáltatóknál elérhető kiszolgálók szoftverkörnyezetét.

1.6.2. Üzembehelyezés

1.6.2.1. Hová rakjuk a szkriptjeinket?

Az Apache szerver alapértelmezett mappája Microsoft Windows operációs rendszeren a következő szokott lenni:

C:\Program Files (x86)\Apache Group\Apache2\htdocs

Ez azt jelenti, hogy itt kell elhelyeznünk a PHP szkriptjeinket.

1.6.2.2. A PHP visszavág

A PHP 4.x-es verzióinak egy sajátosságára szeretnénk felhívni a figyelmet Windows operációs rendszereken. A PHP hibátlan konfiguráció esetén sem hajlandó adatbázist kezelni, amíg végre nem hajtjuk az alábbi SQL parancsot, az idézőjelek közé behelyettesítve a megfelelő adatokat: ezzel lényegében egy jelszó-visszaállítást végzünk el, (megadhatjuk régi jelszavunkat is).

```
SET PASSWORD FOR 'username'@'hostname' = OLD_PASSWORD('password');
```

1.6.3. A PHP és a MySQL tesztelése

Jó gyakorlat, ha a fejlesztőkörnyezetünkre is feltelepítjük a fentebb említett szerveralkalmazásokat, így egy tesztkörnyezetet kapunk. Egy helyi tesztkiszolgálóval gyorsan és kényelmesen dolgozhatunk a fejlesztés során.

Feltételezem, hogy ezen a ponton már fel van telepítve és konfigurálva van mind a MySQl, mind a PHP, mind pedig az Apache szerver.

Teszteljük le először, hogy tudunk-e PHP-t futtatni! Ehhez hozzunk létre egy formázatlan szövegfájlt teszt_PHP.php néven és mentsünk el az Apache webmappájába, azaz ide: C:\Program Files (x86)\Apache Group\Apache2\htdocs

Figyelem: Ehhez a művelethez Windows Vista-tól felfelé rendszergazdai jogosultság szükséges, mivel egy rendszermappa tartalmát módosítjuk!

A teszt_PHP.php PHP szkriptünk tartalma a következő legyen:

```
<?php
print("Működik a PHP.");
?>
```

Nyissunk meg egy böngészőt és gépeljük be a következőt: http://localhost/teszt.php

Nyomjunk ENTER-t és a „Működik a PHP.” szövegnek kell megjelennie a böngészőben. Ha nem jelenik meg semmi, vagy hibaüzenettel szembesülünk, akkor ellenőrizzük, hogy a szkriptet

helyesen gépeltük-e be, ill. az Apache szervermonitor címsorában megjelenik-e a PHP verziója, stb.

Második tesztként ellenőrizzük, hogy hozzáférünk-e PHP szkriptjeinkkel a MySQL adatbázisainkhoz!

A MySQL alapesetben telepít egy 'test' elnevezésű adatbázist, most ezt fogjuk használni. Amennyiben nincs ilyen adatbázisunk, könnyedén létrehozhatunk egyet a CREATE DATABASE *nev*; mysql paranccsal.

A szerveren található adatbázisokat a mysql parancssori használatakor a SHOW DATABASES; paranccsal listáztathatjuk ki.

Most hozzunk létre ismét egy formázatlan szövegfájlt teszt_DB.php néven és mentsük el az Apache webmappájába! A szkript tartalma ez legyen:

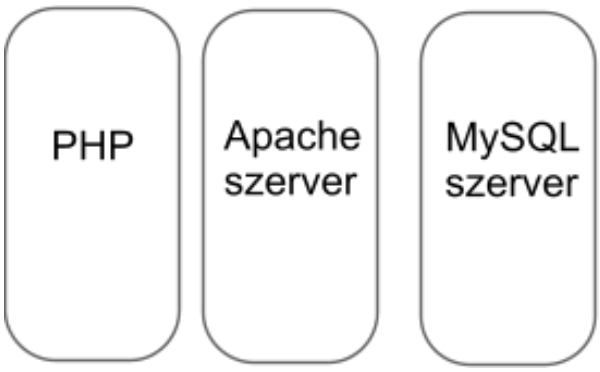
```
<?php  if (!$kapcsolat =
@mysql_connect('localhost',  'username',
'password')) {die(print("Hiba!"));}
if(!mysql_select_db('test',$kapcsolat))
{die(print("Hiba!"));} mysql_close($kapcsolat); print("Adatabázis elérhető.");  ?>
```

A szkript létrehozza a kapcsolatot a MySQL szerverrel és kiválaszt egy adatbázist (a példában a 'test' nevűt), majd lezárja a kapcsolatot. Hiba esetén a szkript kiíratja a „Hiba!” szöveget és befejezi a futást.

Ha mindent jól csináltunk, a böngészőbe beírva az alábbi címet http://localhost/teszt_DB.php az „Adatbázis elérhető.” szöveget kell látnunk.

Ha nem várt eredményt kapunk, ne keseredjünk el, ellenőrizzük ismét, hogy nem gépeltünke el valamit, ill. helyes-e a rendszer konfigurációja!

Tekintsük át rendszerünk kommunikációs rétegeit!



Távoli adatbázis-eléréshez tehát szükség van egy közvetítőrétegre (jelen esetben PHP) és egy webszerverre.

Ezzel a fejlesztéshez és teszteléshez szükséges környezetet felépítettük.

1.7. Fejlesztőeszköz

Egy nagyon fontos összetevő még hátravan: a fejlesztőrendszer. Könyvünk az Adobe Flash Builder integrált fejlesztőkörnyezet használatát mutatja be a mobilalkalmazások fejlesztéséhez. A fejlesztőeszköz bemutatásáról szól a következő fejezet.

2. A Flash Builder

2.1. A Flash Builder és a mobilvilág

Az ActionScript-alapú fejlesztőeszközök két vezető eszköze az Adobe Flash Builder és az Adobe Flash Professional. Előbbi egy Eclipsealapú, komplett fejlesztőrendszer, utóbbi pedig egy erősen dizájnszemléletű, szintén nagyon hatékony fejlesztőeszköz. Könyvünk példáiban a Flash Builder-t fogjuk használni. (Mindkét eszköz fizetős, liszenszköteles szoftver, de lehetőség van néhány napos próbára is.)

A Flash Buildernek jelenleg a következő verziói kaphatóak:

- Flash Builder 4.7 Standard
- Flash Builder 4.7 Premium
- Flash Builder 4.5 for PHP Standard
- Flash Builder 4.5 for PHP Premium

Könyvünkben a Flash Builder 4.7 Standard verziójával fogunk dolgozni.

A Flash Builder-rel robusztus, nagy teljesítményű üzleti alkalmazásokat, professzionális weblapokat, vagy akár játékokat is fejleszthetünk, több más programozási nyelvet és szerveroldali technológiák széles palettáját felhasználhatjuk a fejlesztéseinkhez, köztük a PHP technológiát is.

2.2. A Flash Builder telepítése

A Flash Builder rendszerkövetelményei:

Windows

2GHz-es vagy gyorsabb processzor

Microsoft® Windows® XP Service Pack 3, Windows 7 (32 bit vagy 64 bit), Windows 8
2GB RAM
5GB HDD
Java™ Virtual Machine (32 bit): Oracle® JRE 1.6 vagy 1.7
Java Virtual Machine (64 bit): Oracle JRE 1.6 vagy 1.7
1024x768-as felbontású kijelző (1280x800 ajánlott)
16-bites videókártya
Eclipse™ 3.7 vagy Eclipse 4.2 (a plug-inként történő teleptéshez)

Mac

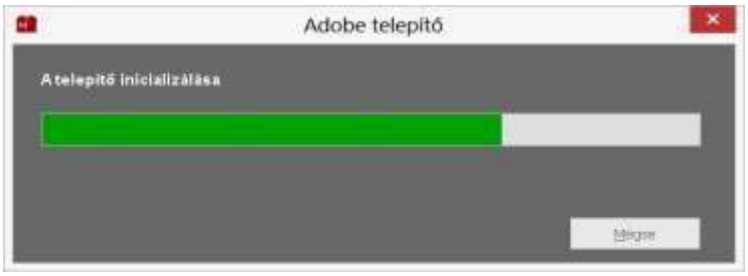
Intel® processor
Mac OS X v10.6, v10.7, vagy v10.8
2GB RAM
4GB HDD
Java Virtual Machine (64 bit): JRE 1.6
1024x768-as felbontású kijelző (1280x800 ajánlott)
16-bites videókártya
Eclipse 3.7 vagy Eclipse 4.2 Cocoa version (a pluginként történő telepítéshez)

A Flash Builder kétféleképpen telepíthető: önálló alkalmazásként, vagy Eclipse pluginként. Utóbbi esetben a következő Eclipse disztribúciók támogatottak: *Eclipse IDE for Java EE developers, Eclipse IDE for Java developers, Eclipse Classic, and Eclipse for PHP developers*.

Az önálló alkalmazásként történő telepítés mindent tartalmaz és gyakorlatilag néhány kattintásból áll. A telepítő 32 és 64 bites változatokban is elérhető, a könyvben a 64 bites Windows változatot fogjuk használni, önálló alkalmazásként telepítve. A telepítő magyar nyelven is tud kommunikálni.

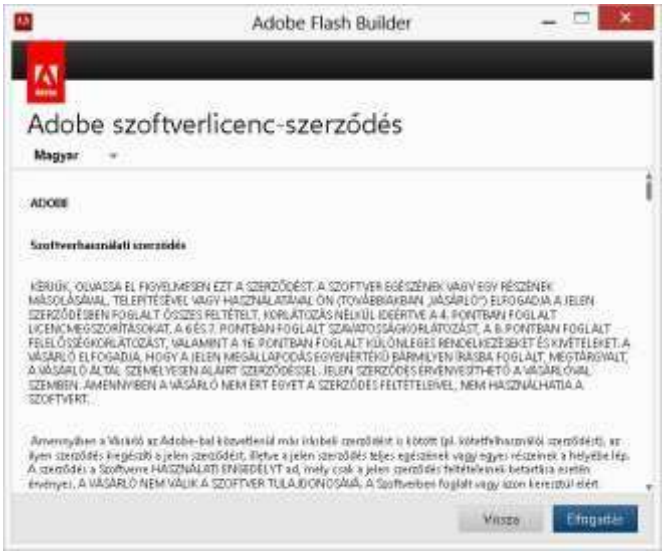
A telepítés a szükséges fájlok automatikus kicsomagolásával kezdődik.

Következő lépésként a telepítő inicializálására kerül sor.



Ezt követően választhatunk, hogy csak kipróbáljuk-e a terméket néhány nap erejéig, vagy végleges (ez utóbbi telepítési módot mutatja be a könyv is) használatra telepítjük. Ez utóbbi esetben elő kell készítenünk a megvásárolt termékhez kapott sorozatszámot is.

Következő lépésként el kell fogadnunk az Adobe szoftverlicenc-szerződését.

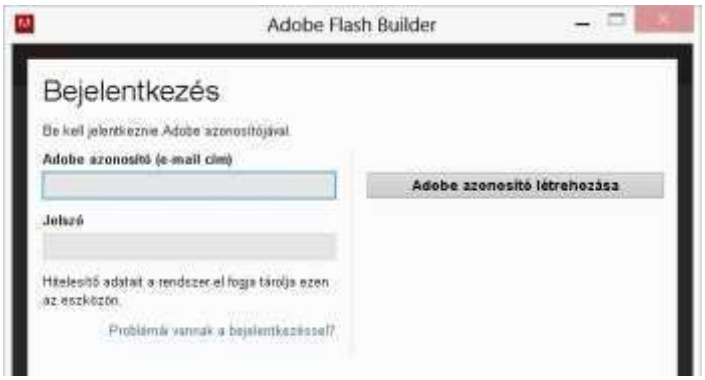


Ezt követően meg kell adnunk a termék egyedi sorozatszámát. Mielőtt a 'Tovább' gombra kattintanánk, ellenőrizzük, hogy aktív internetkapcsolattal rendelkezünk-e, mert a telepítő

interneten keresztül aktiválni fogja Flash Builder példányunkat! A tényleges telepítés addig nem kezdődik el, amíg ez nem történik meg.

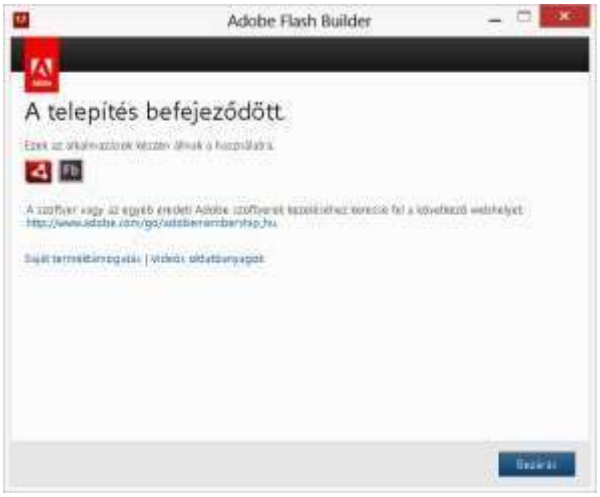
Ezt követően a telepítő felhívja a figyelmet arra, hogy utolsó lépésként meg kell adnunk az Adobe Creative Cloud fiókunk adatait. Ne aggódjunk, ha még nincs ilyen fiókunk. Az Adobe weboldala mellett a telepítővel is létrehozhatunk egyet.

A bejelentkezés gyors, egyszeri és nem igényel további felhasználói interakciót a későbbiekben.



A telepítés következő lépésének beállításait nyugodtan hagyhatjuk az alapértékeken.

A telepítés folyamata kb. 15-20 percnél nem tart tovább.



Immár rendelkezésünkre áll egy komplett androidos fejlesztőrendszer! A Flash Buildert az alábbi ikonnal indíthatjuk el.

2.3. A Flash Builder kezelőfelülete

A Flash Builder az alábbi képernyővel indul:

Az IDE fő részei a következők:

Menüsor és ikonsor (1.) a leggyakoribb műveletek gyors eléréséhez.

Nézetek (2.). A Flash Builder két legfontosabb nézete a Flash és a Flash Debug, mi az előbbivel foglalkozunk majd. Egy nézet nem más, mint a szerkesztőablakok elrendezése. Ezeket az ablakokat a Flash Builder „**view"**knak, a nézeteket pedig „**perspective"**-nek nevezi. A nézetek ablakainak méretét és elrendezését tetszőlegesen módosíthatjuk, akár több ablakot is megnyithatunk, ilyenkor átlapolt elrendezésben jelennek meg az ablakok és fülekkel válthatunk közöttük. Ha egy View fülére duplán kattintunk, akkor a Flash Builder ikonsor alatti részét az adott View fogja teljesen kitölteni. Ha még egyszer duplán kattintunk a fülön, visszaáll az előző méret.

Szerkesztőterület (3.) („**editor"**-ok): Ebben az ablakban szerkeszthetjük alkalmazásunk forráskódjait és itt jelenik meg a kódellenőrzés, a fordítás, illetve a Debug futtatás kimenete is. Szintén itt található az alkalmazás indításakor automatikusan betöltődő 'Welcome Screen' is, ahonnan közvetlenül is létrehozhatunk, vagy megnyithatunk projekteket és számos online információforráshoz is hozzáférhetünk.

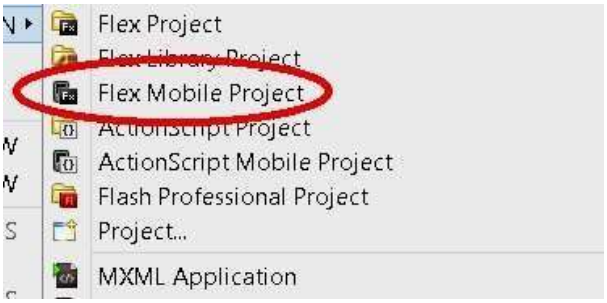
Package Explore (4.): ebben az ablakban projektjeink jelennek meg. Egy projekttérhez („**workspace"**) több projekt is tartozhat. A workspace tulajdonképpen egy komplex mappastruktúra, melyben a projektjeink almappákba rendezve vannak tárolva. Egy új projekt tehát a workspace egy almappájába lesz elmentve és ez jelenik majd meg a Package Explorerben is. A Package Explorer-ben lényegében ezeknek a mappáknak a tartalmát láthatjuk, egy hierarchikus struktúrába rendezve. Tetszőleges Workspace-t létrehozhatunk, de egyszerre mindig csak egy lehet megnyitva. A könyv példáihoz a 'Flash_WP' nevű könyvtárat fogjuk használni a workspace tárolásához. A Flash Builder alapértelmezés szerint a Windows aktuális felhasználójának dokumentum mappái között hoz létre egy workspace mappát és ennek egyegy almappájában a fejlesztett alkalmazás Debug verzióját és a telepítőkészletét (lásd 2.7 és 2.8 fejezeteket). A workspace-ek között a 'File' menü 'Switch Workspace' menüpontjával tudunk váltani, ez a Flash Builder újraindítását igényelheti. Ha egy Workspace-en belül egy projekttel nem dolgozunk aktívan, erőforráskímélés végett bezárhatjuk azt. Ehhez kattintsunk a jobb egérgombbal a Package Explorerben a projekt nevére és a helyi menüből válasszuk ki a 'Close project' menüpontot. A projekt ikonja bezárt állapotban kiszürkített. Bezárt projektet úgy tudunk leggyorsabban újra megnyitni, hogy duplán kattintunk a nevére. A workspace könyvtárak gyökerében létrejön egy '.metadata' nevű könyvtárt is, mely a Workspace-ről tárol különféle leíró adatokat és a Flash Builder kezeli. Soha ne töröljük, vagy módosítsuk kézzel ezt a könyvtárat!

Az **Outline (5.)** nevű ablak az aktuálisan szerkesztett forrásfájl objektumait jeleníti meg. Itt most csak a rend kedvéért említjük meg, a könyvben a későbbiekben nem fogjuk használni.

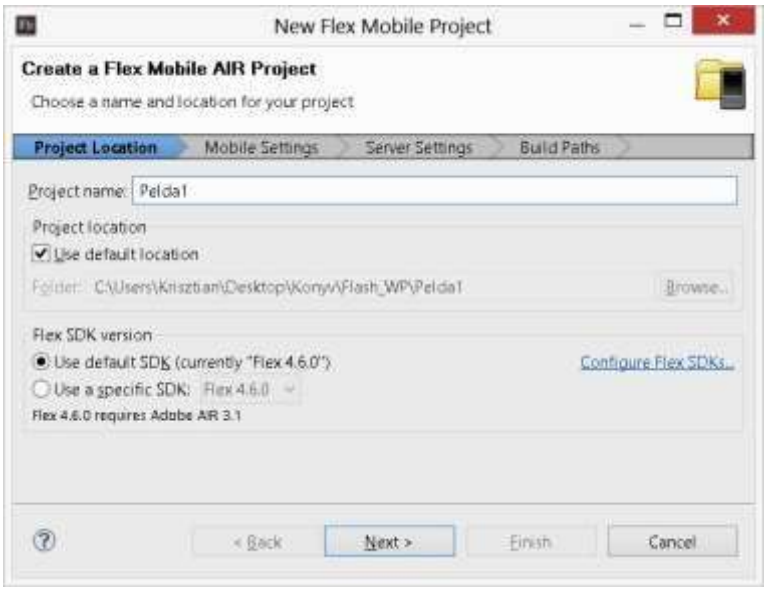
2.4. Az első mobilprojekt létrehozása

A Flash Builder alapvető funkcióinak mélyebb megismerése érdekében hozzunk létre egy mobilprojektet!

Kattintsunk a File menü New, majd 'Flex Mobile Project' pontjára!



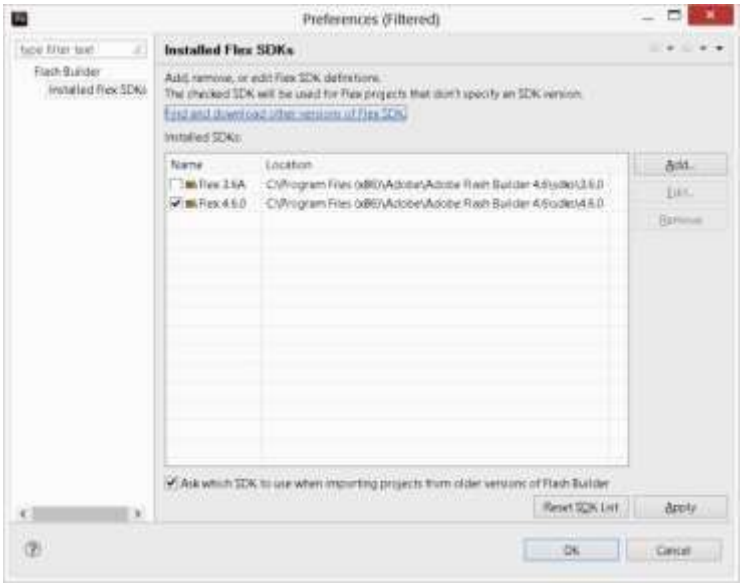
A megjelenő varázsló párbeszédablakában több lépésben be kell állítanunk új projektünk alapvető tulajdonságait. A 'Project Location' lapon először is nevet kell adnunk projektünknek, illetve kiválaszthatunk egy tetszőleges Flex SDK-t. A példaprojektnek a 'Pelda1' nevet adtuk.



2.4.1. Pár szó a Flex SDK-ról

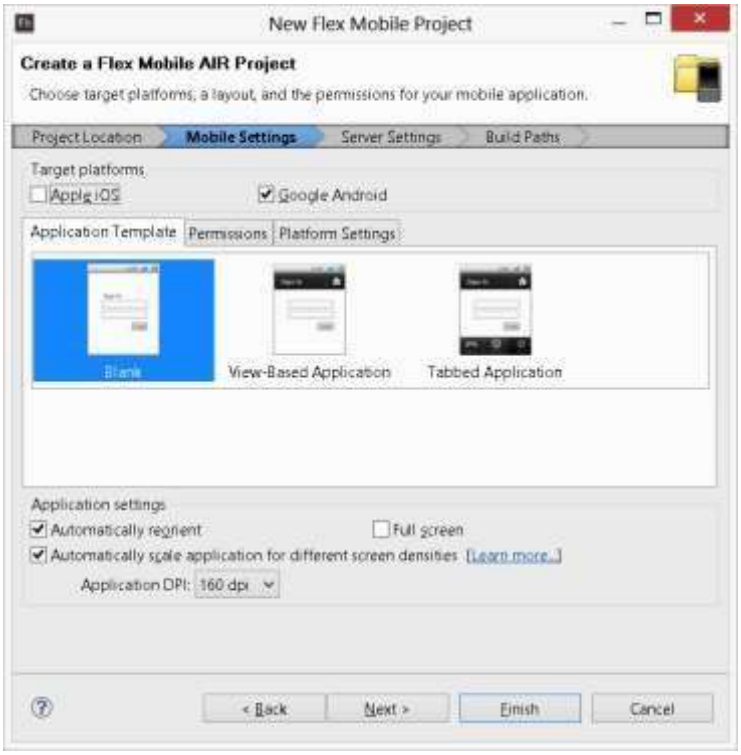
A Flex SDK tulajdonképpen nem más, mint egy könyvtár a számítógépen, mely számos, a fejlesztéshez használható komponenst tartalmaz.

Az aktuális SDK-t az Apache Foundation weboldaláról tölthetjük le (http://flex.apache.org). Szintén innen letölthetünk egy ún. SDK Manager nevű segédprogramot, aminek a segítségével igényeinkhez szabottan, egy lépésben letölthetjük a legújabb SDK-t.



Az imént ismertetett párbeszédablakban a 'Configure Flex SDKs' hivatkozásra kattintva adhatunk meg további SDK elérési útvonalakat a rendszerünkben. A 3.6-os és a 4.6-os SDK-k automatikusan feltelepülnek a Flash Builder rendszerünkkel. A könyv írásakor legfrissebb SDK a 4.11-es.

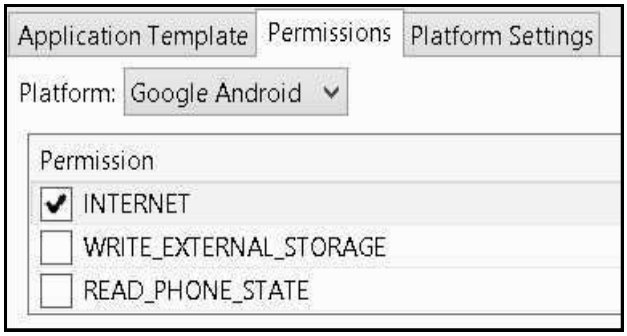
Térjünk vissza a 'New Flex Mobile Project' párbeszédablakhoz és kattintsunk a 'Next' gombra!



A „Mobile settings” lapon a célplatform alapvető tulajdonságait adhatjuk meg projektünk számára. Példánkban egyedül a 'Google Android'-ot válasszuk ki!

Az 'Application Template' fülön válasszuk a 'Blank' sablont, amivel egy teljesen üres felületen, „nulláról indulva” építhetünk fel alkalmazásfelületeket. (A másik két lehetőséggel, a View-alapú és a Tab-alapú alkalmazásfelületekkel a könyv nem foglalkozik.)

A 'Permissions' fülre kattintva megadhatjuk, hogy alkalmazásunk milyen jogosultságokat igényel, azaz az adott mobileszköz mely szolgáltatásaira tart igényt.



A jogosultságokat az Android nagyon komolyan veszi. Amennyiben a megfelelő jogosultság nincs beállítva, úgy a kész alkalmazás akkor sem fog tudni hozzáférni az adott szolgáltatáshoz, ha maga a forráskód hibátlan!

Az egyes jogosultságok jelentése:

INTERNET: internet-hozzáférés engedélyezése.
(AIR alkalmazások esetében kötelező beállítani.)

WRITE_EXTERNAL_STORAGE: hozzáférés a készülékek írható memóriájához, mint fájlrendszerhez.

READ_PHONE_STATE: a telefon állapotának lekérdezése.

ACCESS_FINE_LOCATION: hozzáférés a GPS helymeghatározás által szolgáltatott adatokhoz.

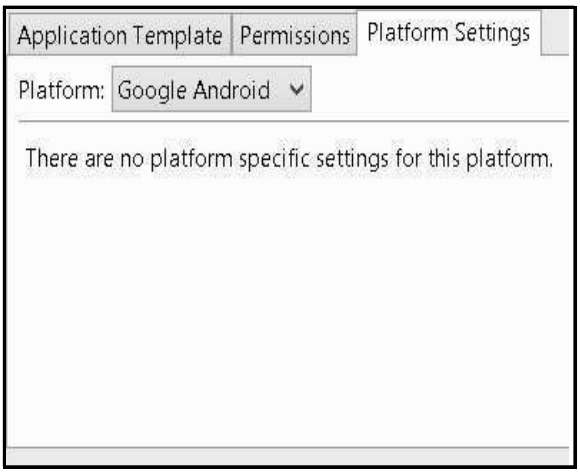
DISABLE_KEYGUARD, WAKE_LOCK: a készülék zárolt állapotának kezelése.

CAMERA: hozzáférés a beépített kamerához.

RECORD_AUDIO: hangfelvételi szolgáltatás igénybevételéhez.

ACCESS_NETWORK_STATE,
ACCESS_WIFI_STATE: a hálózati kapcsolat állapotának a lekérdezéséhez.

A 'Platform Settings' fülön nincsenek további beállítási lehetőségek Android esetében.



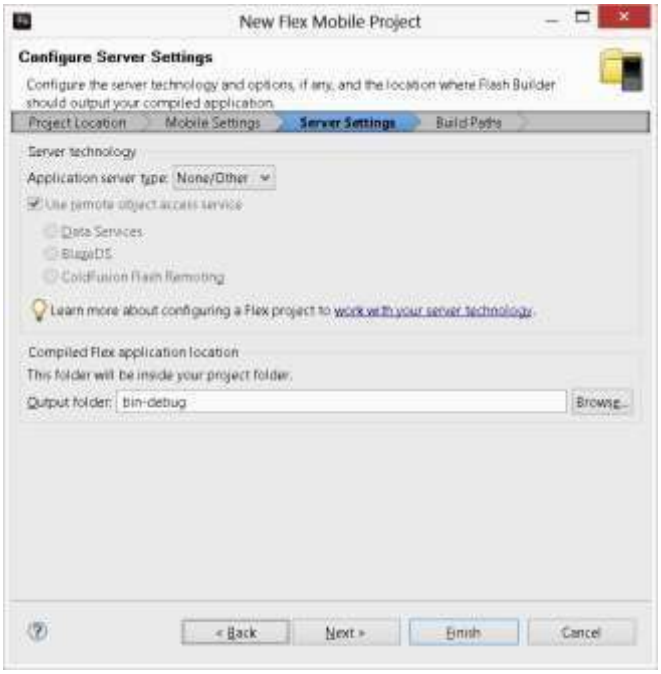
A párbeszédablak alsó részén, az „Application Settings” alatt további beállításokat eszközölhetünk.

'Automatically reorient': bekapcsolva az alkalmazás külön kód írása nélkül is automatikusan megpróbálja áthelyezni a GUI elemeit, ha elforgatjuk a készüléket.

'Automatically scale application for different screen densities': bekapcsolva a különböző DPIvel rendelkező kijelzőknél a megjelenítés automatikusan idomul a készülék aktuális DPIjéhez. Kiválaszthatjuk a viszonyítási DPI értéket is. Az alapérték 160.

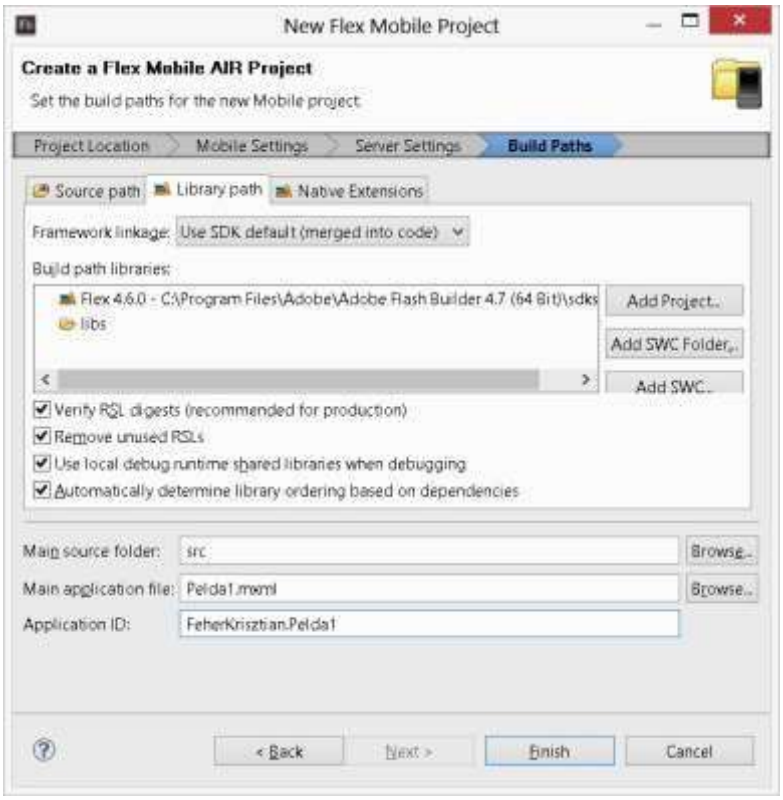
'Full screen': kiválasztva a képernyő alján található szoftveres 'Home', 'Back' stb. gombok is takarásba kerülnek alkalmazásunk felületének javára. Csak nyomós okból érdemes beállítani (pl. teljes képernyős játékok esetében).

A 'Next' gombra kattintva a 'Server settings' lapra jutunk és szervertechnológiák közül választhatunk. A későbbiekben külön kitérünk erre a beállításcsoportra (lásd 5.3 fejezetet).



Első példaalkalmazásunk esetében nyugodtan hagyhatjuk a szervertípust 'None/Other' értéken.

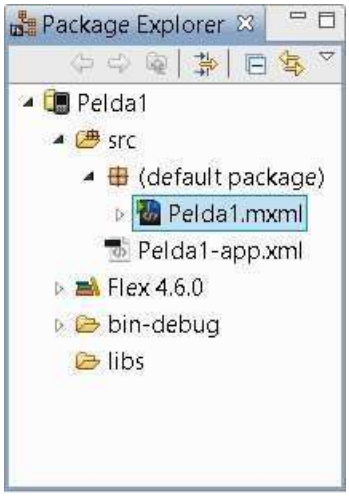
A 'Next' gombra kattintva az utolsó, 'Build Path' lap jelenik meg, ahol fordítással kapcsolatos beállításokat végezhetünk el.



Egyetlen érték változtatását érdemes alapesetben végrehajtanunk ez pedig az Application ID megadása. Ezzel egy egyedi elnevezést biztosíthatunk alkalmazásunk számára. Az alapérték megegyezik projektünk nevével, mi ezt módosítottuk a 'FeherKrisztian.Demo' értékre. Az ID egyes szavait pontokkal kell elválasztanunk egymástól.

Ha végeztünk a beállításokkal, kattintsunk a 'Finish' gombra.

Ekkor létrejön a projektstruktúra a Workspace-en belül, amit a Package Explorer-en keresztül is elérünk. A projektkönyvtár szerkezetét a Windows Intézővel is ellenőrizhetjük.



2.5. Az -app.xml fájlról

A szerkesztőablakban megjelenik a Pelda1.mxml nevű fájl tartalma, mely alkalmazásunk kiindulási forráskódja is egyben. Egy alkalmazás több MXML (Flex), vagy tisztán ActionScript forráskódból is állhat. Könyvünk példái számára bőségesen elegendő egyetlen forrásfájl használata.

Mielőtt a fő MXML forráskóddal megismerkednénk, nézzük meg a 'Package Explorer'-ben szintén látható 'Pelda1-app.xml' fájl tartalmát! Kattintsunk duplán a fájlra! Ekkor az megnyílik a szerkesztőterületen, egy külön fül alatt. (A könyv végén található függelékben egy ilyen XML fájl teljes tartalma áttanulmányozható.)

Az *-app.xml fájl különböző leíró adatokat tartalmaz az adott alkalmazásról, például itt adhatunk meg egyedi ikonokat az alkalmazásnak, beállíthatjuk az adott verziószámot, itt módosíthatjuk utólag is az alkalmazás jogosultságait stb. A fájl szerkezete nem kötelezően fix, adott esetben kézzel is bővíthető.

Az *-app.xml fájl egy XML leírófájl, az egyes adatok <> tag-ek közé vannak beírva. Mindegyikhez <!-- megjegyzés --> alakú XML kommentek formájában rövid leírást is találunk. Amennyiben használni akarunk egy bejegyzést, a megjegyzéstageket el kell távolítanunk.

Ismerjünk meg néhány fontos bejegyzést!

Adobe AIR verzió

Rögtön a második sorban ezt találjuk:

```
<application xmlns="http://ns.adobe.com/air/application/3.1">
```

A bejegyzés az adott Flex SDK-hoz tartozó minimális Adobe AIR verziót adja meg. Ha egy adott Flex SDK-ról átállunk egy másikra, elképzelhető, hogy más minimális AIR verzióval kompatibilis csak. Ilyenkor itt módosítanunk kell a verziószámot. Az XMLNS a NameSpace, azaz névtér rövidítése (lásd következő fejezetet).

Alkalmazásnév és verziószám Módosíthatjuk az alkalmazás ID-ját:

<id>Feher.Krisztian.Demo</id>

Megadhatjuk, hogy mi legyen a fájl neve. A telepítőcsomag fájlneve ez a név, plusz az .APK fájlkiterjesztés lesz:

<filename>Demo</filename>

Az alkalmazás verziószáma. A Google Play Áruház és az Android csomagtelepítője ezt a verziószámot veszi alapul a verziók összehasonlításához. <versionNumber>0.0.0</versionNumber>

Alkalmazás ikonja

A telepített alkalmazásunkhoz egyedi ikonokat rendelhetünk. Csupán az ikonok nevét megadva, azokat a Demo.mxml fájl könyvtárába (ez alapértelmezés szerint a projekt 'src' nevű könyvtára) kell helyeznünk. Nagyon fontos az ikonok méretének betartása, máskülönben az alkalmazásunk nem fogja tudni használni őket! Nem szükséges viszont minden méretben elkészíteni alkalmazásunk ikonját. Az ikonképek ajánlott formátuma a PNG.

```
<icon>
      <image16x16></image16x16>
      <image32x32></image32x32>
      <image36x36></image36x36>
      <image48x48></image48x48>
      <image57x57></image57x57>
      <image72x72></image72x72>
      <image114x114></image114x114>
  <image128x128></image128x128>
</icon>
```

Jogosultságok módosítása

Alkalmazásaink jogosultságait utólagosan az alábbi kezdetű bejegyzések módosításával változtathatjuk meg:

```
<uses-permission android:name="..."/>
```

Egyelőre nem kell beállítanunk semmit az app.xml fájlban, így nyugodtan be is zárhatjuk a szerkesztőablakát.

2.6. Az alkalmazások Flex forráskódja

Most nézzük meg közelebbről hogyan néz ki egy mobilalkalmazás forráskódja! Első példaalkalmazásunk egy tisztán Flex alkalmazás lesz, ActionScript kódot egyelőre nem írunk.

A példaalkalmazás semmi mást nem fog csinálni, mint egy szöveget megjeleníteni a felhasználói felületen.

A Flex kódja egy XML leírófájl. Az összes látható és nem látható komponenst XML tag-ek közé beírva adhatjuk hozzá alkalmazásunkhoz. A Flex komponensei ActionScriptben vannak elkészítve, létrehozásukhoz viszont nem kell egyetlen sor ActionScript kódot sem írnunk. A Flex nagyon rugalmas és könnyen bővíthető, segítségével rendkívül rövid idő alatt létrehozhatunk tetszetős és intelligens felhasználói felületeket.

Új projekt létrehozásakor a Flash Builder automatikusan generál egy kódkeretet, amit akár azonnal le is fordíthatunk.

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<s:Application
xmlns:fx="http://ns.adobe.com/mxml/2009"

xmlns:s="library://ns.adobe.com/flex/spark" applicationDPI="160">

<fx:Declarations>
<!-- Place non-visual elements (e.g., services, value objects) here --> </fx:Declarations>

</s:Application>
```

A második sor a Spark **névteret** definiálja (XMLNS:S). A névtér a modern programozási nyelvek egyik fontos tulajdonsága. Egy névtér a hozzárendelt változókat, függvényeket stb. minősíti, illetve azonosítja más névterekhez képest.

A **Spark** tulajdonképpen a Flex komponenseket tartalmazó architektúra. Rendelkezésre áll egy korábbi architektúra is, az **MX**. Ez utóbbit csak desktop alkalmazásoknál használhatjuk, ott viszont a Spark-kal keverten is. Mobilalkalmazásoknál tehát a Spark komponensek használata lehetséges.

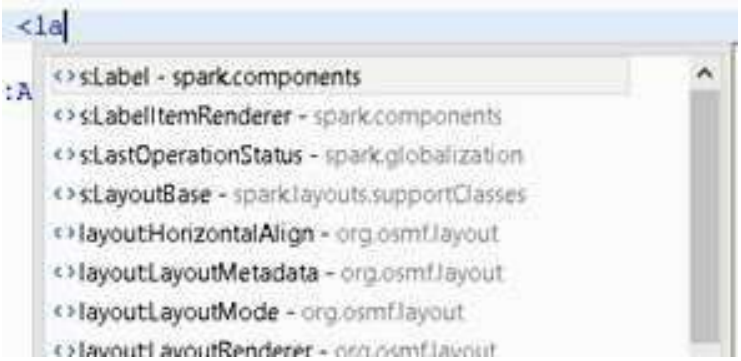
Látható, hogy az egész alkalmazás kódja két <s:Application> tag közé írandó, ahol a „:s" szintén a Spark névteret jelenti.

Az <fx:Declarations> ... </fx:Declarations> tagek közé írt Flex kóddal nem-vizuális komponenseket hozhatunk létre. Könyvünkben ilyen Flex komponenseket nem használunk így ezt a kódrészt akár el is hagyhatjuk a forráskódból.

A Flex nyelvi lehetőségeivel igen összetett viselkedéssel ruházhatunk fel felhasználói felületeket, ám a valódi alkalmazáslogikát ActionScript-ben kell majd megírnunk. (Erre a könyv tényleges példaalkalmazásánál térünk ki a 4. és 5. fejezetben.)

Mielőtt ezt elkészítenénk, helyezzünk el egy szövegcímkét a felhasználói felületen a „Demo alkalmazás." felirattal.

A Flash Builder jelentősen megkönnyíti a kódbeírást az ún. „Code assist" szolgáltatásával, mely intelligens kódjavaslatok gyors beszúrását teszi lehetővé, automatikus kódkiegészítéssel. Ha nem jelenik meg automatikusan, akkor gépelés közben nyomjuk meg a CTRL+SPACE billentyűkombinációt. Ekkor az éppen beírt komponenshez stb. kapcsolódó tulajdonságokat stb. jeleníti meg egy lebegő listában. A lista mellett részletes leírást is találunk az adott listaelemről. Az <s:La szöveg beírása után nyomjuk meg a CTRL+SPACE billentyűkombinációt és válasszuk ki az <>s:Label javaslatot! A listában a számítógép billentyűzetének fel-le billentyűivel lépkedve, majd az ENTER-t megnyomva, vagy az egérrel duplán kattintva szúrhatjuk be a megjelenített javaslatot.



Közvetlenül az </s:Application> tag feletti sorba írjuk be a következőt:

<s:Label x="20" y="30" text="Demo alkalmazás."/>

A komponensek megadásának fenti módja a rövidített írásmód. Létrehozhattuk volna a komponenst egy hosszabb írásmóddal is:

<s:Label>
<s:x>20</s:x>
<s:y>30</s:y>
<s:text>Demo alkalmazás.</s:text>
</s:Label>

Mindkét írásmód helyes, a használat egyéni ízlés kérdése.

Ezzel elhelyeztünk egy szöveget az alkalmazás képernyőjén, a vízszintes 20. és a függőleges 30. abszolút pixel koordinátákra.

2.7. Az első futtatás

A Flash Builder két lehetőséget is kínál fejlesztett alkalmazásaink futtatására.

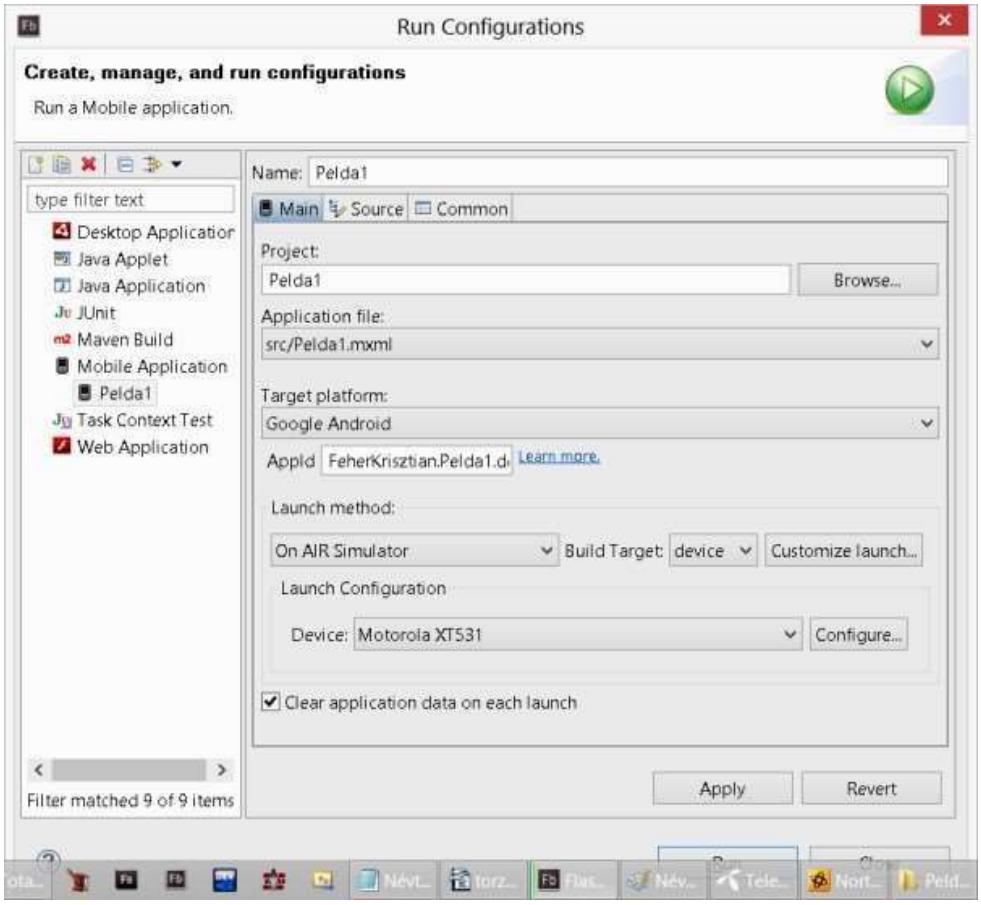
Amennyiben nem rendelkezünk androidos mobileszközzel, fejlesztőrendeszerünk operációs rendszerén is szimulálhatunk egy ilyen eszközt. Ilyenkor az alkalmazás egyszerűen egy ablakban fut.

A Java-alapú fejlesztésekhez hasonló emulált Android rendszer használatára a Flash Builderrel is lehetőség van. Erről a lehetőségről részletesen *Véronique Brossier: Developing Android Applications with Adobe AIR* c. könyvében olvashatunk. (Lásd a könyv végén található irodalomjegyzéket.)

Androidos eszköz megléte esetén az alkalmazásunk ún. debug (a fejlesztés alatt használt, tesztelésre szánt) verzióját közvetlenül az eszközön is futtathatjuk. Ez egy nagyon kényelmes szolgáltatás: az alkalmazás ebben az esetben fel is települ az eszközre, sőt, egy esetleges korábbi verziója el is távolítódik. Mindez automatikusan a háttérben történik meg. Ennek a futtatási módnak az előfeltétele, hogy az eszközt a fejlesztőrendszerünk operációs rendszere fel tudja ismerni. (A saját tapasztalat azt mutatja, hogy egy a készülékbe behelyezett külső SD kártya esetén jóval nagyobb eséllyel ismeri fel az operációs rendszer a készüléket.)

Alkalmazásunk futtatásához kattintsunk a Flash Builder ikonsorának a lejátszás gombot mintázó ikonjára:

Projektünk első futtatásakor a 'Run Configurations' párbeszédablak jelenik meg. Itt be kell állítanunk néhány futtatási tulajdonságot.



A legtöbb esetben csak a 'Main' fül alatt található néhány beállítást használjuk.

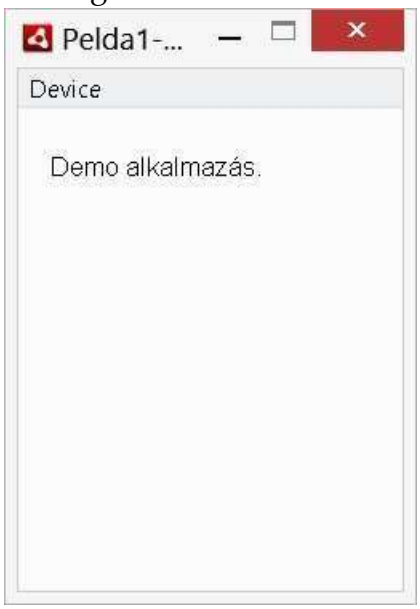
A 'Launch method' lehetőségnél az „On AIR Simulator” lehetőség jelenti a szimulált futtatást. Ilyenkor ki kell választanunk egy készülékprofilt. A Flash Builder-rel együtt feltelepül számos profil, de teljesen egyedit is létrehozhatunk. Egy profil lényegében egy eszköz képernyőtulajdonságait határozza meg, egyedi névvel ellátva.

Az „On device” lehetőség a fentebb említett, közvetlen mobileszközön történő futtatást jelenti. A Debug verzió teljesértékű alkalmazás ugyan, ám csak a fejlesztés alatt érdemes használni, ugyanis fordítás során rengeteg metaadat is beépül az alkalmazásba, melyekkel a hibák keresése végezhető el hatékonyan. A Debug verzió továbbá egy ún. ideiglenes fejlesztőipublikálási tanúsítványt tartalmaz, ami egy idő után (kb. egy év) elévül. A tanúsítványokról a következő fejezetben még szó lesz.

Érdemes bejelölni a 'Clean application data on each launch' lehetőséget. Ilyenkor minden futtatás előtt törlődnek a korábban ideiglenesen tárolt futtatási információk.

A 'Run' gombra kattintva egy párbeszédablak jelenhet meg, mely a forrásfájl mentésére kérdez rá futtatás előtt. Ebben az esetben érdemes kipipálni az 'Always save resources before launching' opciót, hasonló esetben legközelebb így nem fog megjelenni a párbeszédablak.

Az alábbi képen a Windows-on szimulált verzió futása látható. Az alkalmazás felületén megjelenik a fentebb megadott szöveg.

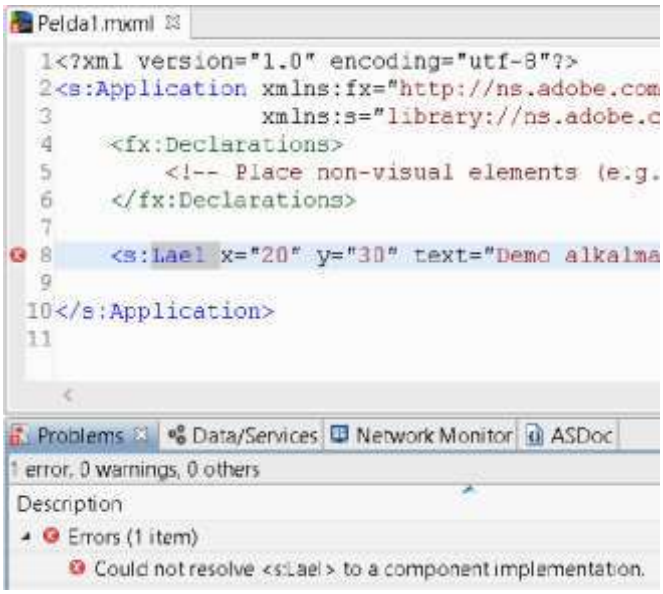


Szimulált futás esetén a szimulációs alkalmazásablakban találunk egy 'Device' menüt. Ennek a menüpontjaival a készülékeken a legtöbb esetben valamilyen fizikailag is megtalálható gombra történő reakciót tesztelhetjük, illetve a különböző irányokba történő forgatást is „szimulálhatjuk”.

A szimulációs mód minősége igen jó, akár úgy is megírhatunk egy komplett alkalmazást, hogy az előtte valódi mobileszközt „látott volna”.

Itt hívjuk fel a figyelmet a Flash Builder kódszerkesztő területének alján található 'Problems' fülre. Itt egy listában jelennek meg a fordítás során talált hibák. A lista a Flash Builder alapértelmezett beállításai mellett az adott állomány vagy projekt minden mentésekor frissül, mivel a Flash Builder alapesetben minden mentéskor kódelemzést és bizonyos fordítási műveleteket is elvégez annak érdekében, hogy minél gyorsabban futtatni tudjuk alkalmazásainkat.

Vizsgáljuk meg, hogyan működik ez a funkció! Ejtsünk szándékosan hibát a kódban! Ehhez módosítsuk a <s:Label kódrészt <s:Labl -re és mentsük el a projektet! A hibára a Flash Builder felhívja a figyelmünket a listában, de a forráskód érintett sora elé is elhelyez egy piros figyelmeztető jelzést:



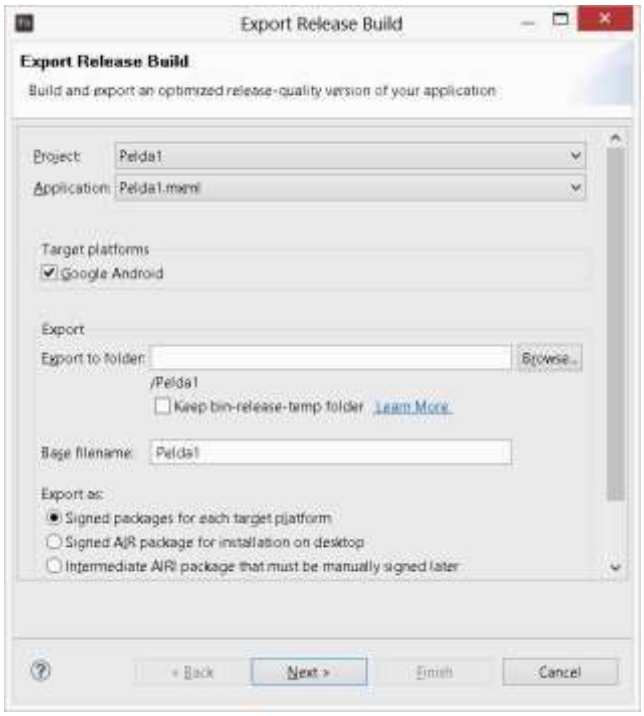
Amennyiben több hibát is talál a kódellenőrzés és hosszabb forráskóddal dolgozunk, a legkönnyebben úgy tudunk a hibás sorra ugrani, hogy a megjelenített hibalistában duplán kattintunk a konkrét hibabejegyzésen.

Most javítsuk ki a hibát és mentsük el a projektünket!

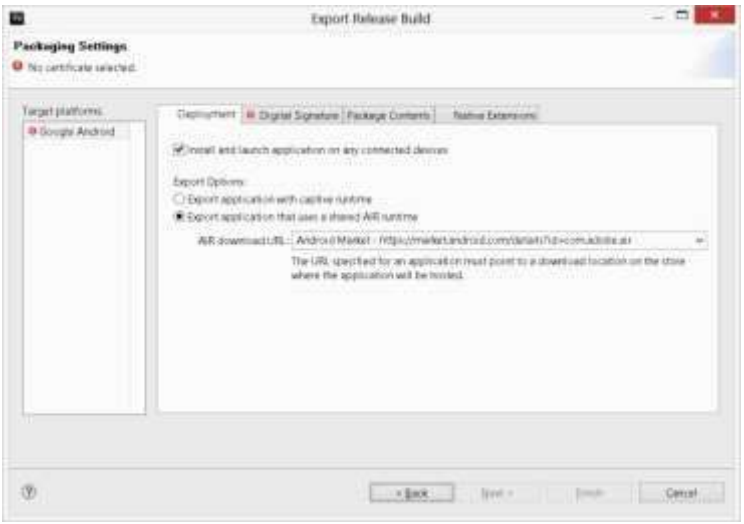
2.8. Kész alkalmazások publikálása

A fejlesztési tevékenység egyik fontos mozzanata a kész alkalmazás telepítőcsomagjának elkészítése, hogy ezzel mások is használhassák alkalmazásunkat.

Ehhez válasszuk ki a 'Project' menü 'Export Release Build' menüpontját! A megjelenő párbeszédablakban beállíthatjuk a publikálás tulajdonságait.



Az 'Export to folder' és a 'Base filename' mezők a legfontosabbak. Előző annak a mappának az elérési útját adja meg, ahová a telepítőcsomagot el akarjuk menteni, utóbbi pedig a kész telepítőcsomag fájlneve lesz. Android esetében egyetlen fájl jön létre, .APK kiterjesztéssel. A 'Next' gombra kattintva további csomagolási beállításokat végezhetünk el ('Packaging Settings').



Az 'Install and launch application on any conected devices' lehetőséget kiválasztva publikálás után egyből telepíthetjük is az alkalmazást a számítógéphez csatlakoztatott eszköz(ök)re.

A 'Deployment' lapon két exportálási mód között választhatunk, melyek azt befolyásolják, hogy a telepített alkalmazás milyen módon használja az Adobe AIR futtatókörnyezetet.

Az 'Export application with captive runtime' csomagolással egy olyan alkalmazást telepíthetünk, melyhez egy komplett Adobe AIR futtatókörnyezet van „hozzácsomagolva". Ebben az esetben a céleszközön nem kell külön telepíteni az Adobe AIR-t. Ez a csomagolási módszer némileg növelheti a telepítőfájl méretét.

Az 'Export application that uses a shared AIR runtime' lehetőséget kiválasztva az alkalmazás igényelni fog egy külön telepített AIR környezetet. Amennyiben a céleszközön nincs ilyen telepítve, a telepítő automatikusan fel fogja ajánlani és lehetővé teszi az Adobe AIR telepítését.

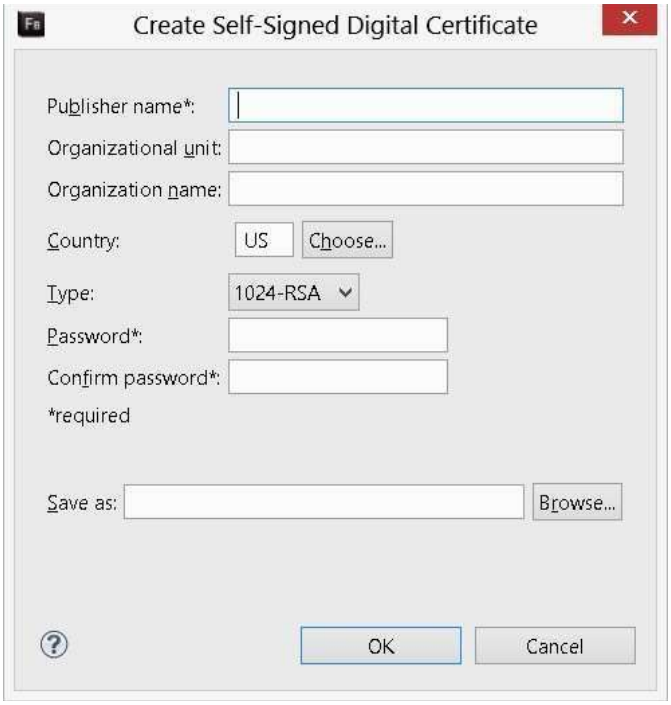
Első exportáláskor a 'Digital Signature' lapnál egy piros figyelmeztető jelzést találunk. Minden androidos telepítőcsomaghoz ugyanis egy fejlesztői tanúsítványt is társítani kell. Ez a fejlesztők azonosítását hivatott biztosítani.

Két lehetőségünk van. Igénylünk tanúsítványt, vagy saját aláírásút készítünk. Mindkettő út járható és teljesen egyenértékű az alkalmazás publikálása szempontjából. Előbbi előnye a nagyobb elfogadottság, utóbbié a gyorsaság, hiszen azonnal és nem mellesleg ingyen

előállíthatunk egyet a Flash Builder-rel (a külső féltől származó tanúsítványok beszerzése költségekkel járhat). A tanúsítványokhoz jelszót is meg kell adnunk.

Maga a tanúsítvány egy fizikailag is különálló titkosított fájl, amit meg kell adnunk az első exportálás során, a hozzá tartozó jelszóval együtt. A létrehozható tanúsítványok száma nincsen korlátozva.

A visszaélések elkerülése érdekében tanúsítványainkat soha, semmilyen körülmények között se adjuk oda másnak és ne tegyük közzé!



Ha ezzel elkészültünk, visszatérhetünk a csomagolásbeállító párbeszédablakhoz. A 'Package Contents' és a 'Native Extensions' lapok beállításait nem kell módosítanunk.

A 'Finish' gombra kattintva a Flash Builder elkészíti a megadott könyvtárban a telepítőcsomagot.

A telepítőcsomagot feltölthetjük a 'Google Play', vagy az 'Amazon Appstore for Android' webáruházakba.

Az alkalmazások webáruházban történő közzététele regisztrációhoz kötött.

A Google Play Áruházban egy egyszeri regisztrációs díj (kb. 25 dollár) kifizetése és a megfelelő szerződési feltételek elfogadása után ingyenesen, vagy díj ellenében letölthetővé tehetjük alkalmazásainkat. A fejlesztői regisztráció egy meglévő Google fiókkal lesz összekapcsolva és külön adminisztrációs felületen végezhetjük el az alkalmazásaink közzétételét, ill. frissítését.

Elméletileg semmilyen megkötés nincsen arra nézve, hogy hol szabad az elkészült alkalmazásokat közzétenni. Akár saját honlapunkon is elérhetővé tehetjük azt! Ebben az esetben viszont egy androidos eszközről nem tudjuk közvetlenül feltelepíteni őket. Előbb le kell töltenünk asztali számítógépen egy adathordozóra (pl. memóriakártyára, amit utána behelyezünk a mobilkészülékbe) és manuálisan kell elindítanunk a telepítést. Ezzel elkészültünk első alkalmazásunkkal.

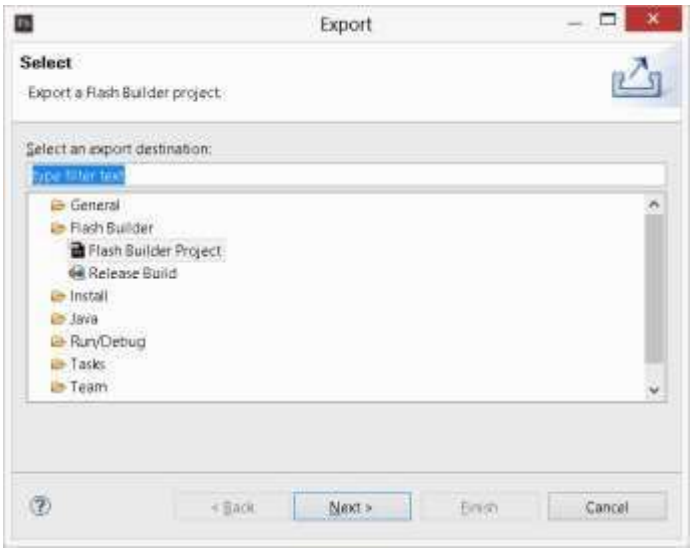
A következő fejezetekben az imént bemutatott lépéseket már nem fogjuk újra leírni, csupán hivatkozunk majd rájuk.

2.9. Projektek exportálása, importálása

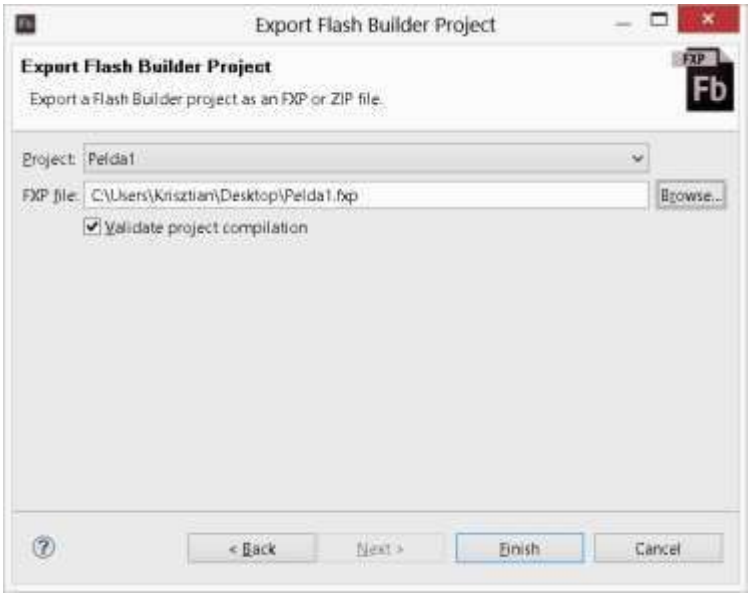
A Flash Builder-rel készített projektek mozgatásának egyik egyszerű módja az exportálás/importálás, mely mindig az aktuálisan használt workspace-re vonatkozik.

Az aktuális projektet a 'File' menün keresztül is exportálhatjuk, de alternatívaként a Package Explorer-ben a jobb egérgombbal a projekt nevére kattintva a helyi menüből az 'Export' lehetőséget is kiválaszthatjuk.

Ekkor egy varázsló indul el. Az első párbeszédablakban ki kell választanunk a projekt típusát. Esetünkben ez egy Flash Builder projekt.

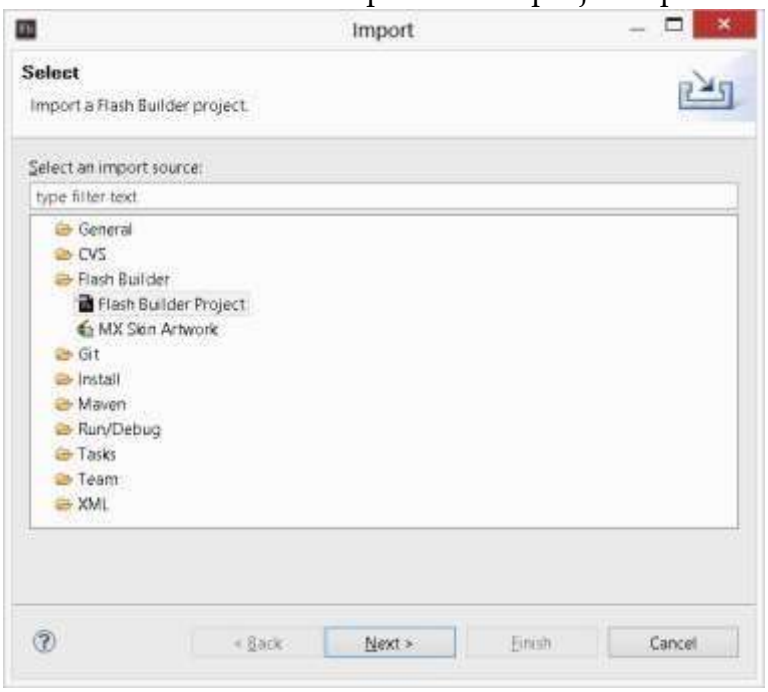


A 'Next' gombra kattintva a következő párbeszédablakban mégegyszer pontosíthatjuk, mely projektet akarjuk exportálni és meg kell adnunk egy célkönyvtárat is.

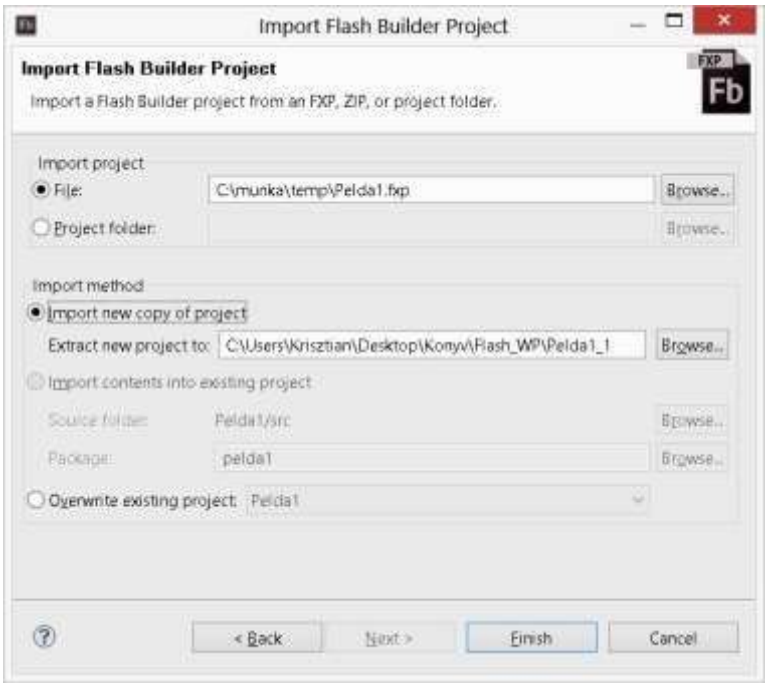


A 'Finish' gombra kattintva megtörténik az exportálás. A megadott könyvtárban egy .FXP kiterjesztésű fájl jön létre.

Projektek importálása hasonlóan történik. Az importálást végző varázsló első párbeszédablakában ki kell választanunk az importálandó projekt típusát.



A következő lépésben pedig megadhatjuk az importálandó projekt helyét, valamint azt is, hogy milyen módon jöjjön létre az új projekt az aktuálisan használt workspace-ünkben.



Amennyiben már létezik egy, az importálandó projekttel megegyező nevű projekt, az importált projektet átnevezhetjük, így az új projektként fog megjelenni.

Egy meglévő projektbe is importálhatunk, illetve megegyező név esetén felül is írhatjuk az importálandó projekttel a régit.

A 'Finish' gombra kattintva végrehajtódik az importálás.

3. Flex/ActionScript alapismeretek

3.1. ActionScript

Az ActionScript jelenlegi legfrissebb verziója a 3.0. A könyvben az Actionscript részletes ismertetésére nem vállalkozunk, de a nyelv legfontosabb alaptulajdonságait mindenképpen meg kell ismerni.

3.1.1. Adattípusok

Minden program adatokkal dolgozik, melyeket szimbolikus nevekkel látunk el. Az adatok tartalma a program futtatása során megváltoz(hat)nak, ezért **változók**nak nevezik őket. Többféle típusú változó áll a programozók rendelkezésére, ezeket a típusokat **adattípusoknak** nevezzük.

Az ActionScript 3.0-ban az adattípusok mindegyike objektumot takar. Kétféle adattípuscsoportot különböztetünk meg: **primitív** és **komplex** adattípusokat. Az egyes adattípusok nevei, lehetséges értékei és közvetlen **deklaráció** (létrehozás) utáni alapértelmezett értékei a következők.

Primitív adattípusok

Boolean: csak két értéket vehet fel. true vagy false. Az alapértelmezett érték a false.

int: 32 bites előjeles egész számok tárolására az 2,147,483,648 (-2^{31}) - 2,147,483,647 ($2^{31} - 1$) értéktartományban. Az alapértelmezett érték 0.

uint: 32 bites előjel nélküli egész számok tárolására használható a 0 és 4,294,967,295 (232 – 1) értéktartományban.

Null: csak egyetlen értéket vehet fel, ez pedig a null. Fontos megjegyezni, hogy a komplex és a String adattípusok alapértelmezett értéke szintén null. A Boolean, int, Number, uint adattípusok viszont nem vehetik fel a null értéket, amennyiben null értéket adunk ilyen változóknak, az ActionScript egy megfelelő alapértelmezett értékkel helyettesíti a null-t (például '0'-val).

Number: 64 bites, dupla pontosságú számok tárolására alkalmas a 4.940656458412467e324 és 1.79769313486231e+308 értékhatárok között. Egész és lebegőpontos számok tárolására is alkalmas, előbbiek esetében akkor javasolt, ha a 32 biten ábrázolható értéktartomány nem elegendő. Alapértelmezett értéke NaN, ami nem definiált értéket jelent.

String: 16 biten, UTF-16 formátumban tárolt szövegek tárolására alkalmas. Alapértelmezett értéke a null, ami nem összekeverendő az üres sztringgel (””).

Object: az osztálydefiníciók alaposztályaként funkciónál. Alapértéke a null.

Void: kizárólag visszatérési értékekhez használatos és egyetlen értéket vehet fel: undefined.

Komplex adattípusok

A komplex adattípusokkal a könyvben nem foglalkozunk, de a rend kedvéért ezeket is megemlítjük.

Array, Date, Error, Function, RegExp, XML, XMLList. Változókat az alábbiak szerint hozhatunk létre: **var** valtozo_neve:típus; például: **var** Szoveg:String;

Az ActionScript ún. case-sensitive szintaktikával rendelkezik, ami azt jelenti, hogy megkülönbözteti a kis- és nagybetűket. A fenti példában deklarált változóra tehát csak 'Szoveg' néven hivatkozhatunk programjaink forráskódjában, a 'szoveg' név használata hibaüzenetet eredményezne fordításkor.

3.1.2. Utasítások

Az ActionScript utasításait pontosvesszővel zárjuk. Az önmagában írt pontosvesszű egy üres utasításnak felel meg. Használható akkor, amikor az adott kódhelyen utasítása megadása kötelező, de nem akarunk semmilyen explicit műveletet végrehajtani.

Az utasításokat blokkokba is foglalhatjuk, ilyenkor több utasítás is egyetlen utasításként lesz végrehajtva. Az utasításblokk általános formája:

```
{
  utasítás1;   utasítás2;
}
```

A záró kapcsos zárójel után nem kell pontosvesszőt írni.

3.1.3. Megjegyzések a kódban

A programozás során hasznos, ha emlékeztetőket, megjegyzéseket, magyarázatokat adhatunk a forráskódhoz. Ezeknek a megjegyzéseknek, kommenteknek nincsen hatása a program működésére, ám saját magunk számára, vagy bárkinek, akinek értelmeznie kell a forráskódot, nagy segítséget jelenthetnek ezek a magyarázó leírások.

Egysoros és kétsoros magyarázatokat használhatunk ActionScriptben, melyeket a // és /* */ jelekkel vezethetünk be.

Rövid példával illusztrálva:

```
// egysoros megjegyzés
/*
    többsoros
    megjegyzés
*/
```

3.1.4. Operátorok és precedenciák

Az alábbi táblázat az ActionScript operátorait tartalmazza csökkenő preferencia-, azaz végrehajtási sorrendben (tehát legfelül a legmagasabb precedenciával rendelkező operátorok állnak).

Csoport	**Operátor**
Elsődleges	[] {x:y} () f(x) new x.y x[y] <></> @ :: ..
Postfix	x++ x--
Unáris	++x --x + - ~ ! delete typeof void
Multiplikatív	* / %
Additív	+ -
Bitenkénti eltolás	<< >> >>>
Összehasonlító	< > <= >= as in instanceof is
Egyenlőség	== != === !==
Bitenkénti ÉS	&
Bitenkénti KIZÁRÓ VAGY	^
Bitenkénti VAGY	|
Logikai ÉS	&&
Logikai VAGY	||
Feltételes	?:
Hozzárendelés	= *= /= %= += -= <<= >>= >>>= &= ^= |=
Vessző, felsorolás	,

Zárójelek alkalmazásával az alapértelmezett precedenciasorrendet más nyelvekhez hasonlóan az ActionScriptben is befolyásolhatjuk.

3.1.5. Elágazási szerkezetek

Különböző feltételek, értékek vizsgálata alapvető programozási gyakorlat. A vizsgálatok eredményétől függően lehetséges más-más műveleteket végrehajtatni. Ennek az eszközeivel is megismerkedünk.

3.1.5.1. if .. else

A feltételek vizsgálatának általános szerkezete:

if (feltétel) utasítás1; else if (feltétel) utasítás2; else utasítás3;

Az if .. else vizsgálatnak minimum egy if -el bevezetett feltételvizsgálatot tartalmaznia kell, ám az else if és else ágak opcionálisak.

Az else if után álló utasítás akkor hajtódik végre, ha a megelőző if feltétele nem teljesül.

Amennyiben csak else-t írunk, az egyfajta alapértelmezett műveletként is felfogható, ami akkor hajtódik végre, ha egyetlen megelőző if, vagy else if feltétel sem teljesül. Az alábbi példa egy vizsgaértékelő program része is lehetne:

```
if (pontszam > 60)
      {
vizsgajegy=5;
  dicseret=true;
     }
 else if ((pontszam <= 60)&&(pontszam >40))
vizsgajegy=4;   else    vizsgajegy =3;
```

3.1.5.2. switch

Amennyiben **ugyanannak a változónak több lehetséges és konkrét** értékeit akarjuk megvizsgálni, akkor az if .. else vizsgálatok nehezen átláthatóvá válhatnak.

A switch használatával elég a vizsgált változót egyszer megadni. Általános formája:

```
switch(valtozo)
{
 case ertek1:   utasitas1;   break;  case ertek2:   utasitas2;
utasitas3;  break; default:  utasitas1;
 break;
}
```

A vizsgált értékeket a case kulcsszóval kell bevezetnünk. A feltétel teljesülése esetén a case után megadott utasítások hajtódnak végre, egészen a break kulcsszóig. A break használata nélkül az összes soron következő case utáni utasítás végrehajtódna.

A default után is ajánlott használni a break-et, azon egyszerű oknál fogva, hogy a default bárhol állhat, nem csak az utolsó helyen.

Ismét egy egyszerű példával élve:

```
switch(osztalyzat)
{
 case 5:
      bizonyitvany=true;        dicseret=true;        break;
    default:
      bizonyitvany=true;        dicseret=false;
  break;
}
```

3.1.6. Ciklusok

Utasítások ismételt végrehajtására ciklusokat használunk. A ciklusokban elhelyezett utasítások legtöbbször bizonyos számú alkalommal, ill. egy bizonyos feltétel teljesüléséig hajtódnak végre, újra és újra. Adott feladat esetén az is cél lehet, hogy a ciklusok meghatározatlan ideig végrehajtódjanak, ekkor ún. végtelen ciklusról beszélünk.

A ciklusokból a meghatározott feltétel teljesülésétől függetlenül a break; utasítással bármikor kiléphetünk.

3.1.6.1. Előltesztelő ciklus (while)

A while ciklust szokás előltesztelő ciklusnak is nevezni, mivel a ciklus végrehajtásának feltétele a ciklus első végrehajtása előtt mindenképpen kiértékelődik.

```
Általános formája: while (feltétel)
{
utasítás;
}
```

Az alábbi példában egy egész számként deklarált i nevű változó értékét annyiszor növeljük, amíg értéke kisebb mint 10.

```
i=0; while (i<10)
{
   ++i;
}
```

3.1.6.2. Hátultesztelő ciklus (do .. while)

A do .. while ciklust szokás hátultesztelő ciklusnak is nevezni, mivel a ciklus végrehajtásának feltétele először csak a ciklus első végrehajtása után értékelődik ki. Általános formája:

```
do
{
utasítás; } while (feltétel)
```

Az előző példa do .. while ciklussal:.

```
i=0; do {
++i;
} while (i<10)
```

3.1.6.3. Számláló ciklus (for)

A for ciklus az elöltesztelő ciklusnak egy speciális fajtája, mivel külön megadható a ciklus ismétlődését befolyásoló változó és a változó léptetésének az értéke. Általános alakja:

```
for( értékadás ; feltétel; léptetés)
{
 utasítás;
}
```

Az alábbi példa 0-tól 100-ig terjedően a páros számok összegét egy szam nevű változóban tárolja el:

```
for(i=0; i<=100;i+=2)
{
 szam+=i;
}
```

A for ciklusok használatakor előfordulhat, hogy bizonyos feltételek fennállása esetén nem akarjuk az egész ciklust végrehajtani, hanem az aktuális műveletnél a ciklus elejére ugrunk, tehát átugrunk bizonyos utasításokat. Erre használhatjuk a **continue;** utasítást. Például:

```
for(i=0; i<=100;i+=2)
{//a 10 és 20 közötti számokat kihagyjuk  if(i>10 && i<20) continue;  szam+=i;
}
```

3.1.7. Függvények

Amennyiben egy több utasításból álló kódblokkot a program több részén is végre akarunk hajtani, érdemes ún. függvényt írni, mellyel az utasításblokkot egyetlen, egyedi, általunk megadott névvel hívhatjuk meg. Létrehozásának általános alakja:

```
 hozzáférés fuggveny_név (argumentumok):visszatérési érték
{
 utasítások;
}
```

Az alábbi példában egy numerikus, szam nevű váltózó értékét egy függvénnyel akarjuk nullázni. A függvény kódja:

```
var szam:int;
function nullaz():void
{
 szam=0;
}
```

A függvény nem ad vissza értéket, ezért típusa void.

Valahányszor csak használni akarjuk a függvényt, elég a nullaz(); utasítást elhelyeznünk programunk megfelelő pontján.

Más visszatérési érték esetén a függvény kódblokkjában elhelyezett return kulcsszóval tudjuk megadni a visszatérési értéket.

Két szöveges változó tartalmának összeakapcsolását például a következő függvénnyel végeztethetjük el.

```
function str_osszaad (s1:String,s2:String):String
{
 return s1+s2;
}
```

A függvényt például így is használhatjuk programunkban:

```
var szoveg1:String; var uj_szoveg:String; szoveg1="Az ActionScript";
...
uj_szoveg=str_osszead(szoveg1,"a legjobb!");
```

Az uj_szoveg nevű változó tartalma a függvény meghívása után ez lesz: "Az Actionscript a legjobb!"

3.1.8. Hozzáférések szabályozása

A gyakorlatban mind a változók, mind a függvények és osztályok hozzáférhetőségét szabályozni szoktuk és szabályozni is kell. (A függvények változóihoz nem kell megadnunk a hozzáférés módját.)

Három hozzáférési módot említek meg: **public:** minden hívó számára hozzáférhető. **protected:** csak a definiáló osztály példányai számára elérhető. **private:** csak a definiáló osztály számára hozzáférhető.

Például:

```
public var szam:int; public function nullaz():void
```

3.1.9. Osztályok

Az objektumorientált programozás alapvető eszközét jelentik az osztályok. Egy osztály funkcionálisan összetartozó függvények és változók egybecsomagolt, absztrakt egysége. Lássunk egy példát:

```
public class osztalynev {        public var valtozo:int;        public function
osztalynev() {}   public function
   muvelet():void
        {
                ++valtozo;
        }
   }
   public uj_osztaly = new osztalynev();
```

A fenti példában létrehozunk egy osztalynev nevű osztályt. Az osztálynak van egy saját változója, azaz **tulajdonsága** és két függvénye, azaz **metódusa**.

A muvelet() nevű függvény a valtozo nevű osztálytulajdonságon végez el műveletet.

A másik függvény neve megegyezik az osztály nevével. Ez egy speciális függvény, az ún. konstruktor, mely az osztálypéldány létrehozásakor automatikusan végrehajtódik. Különféle inicializációs műveletek végrehajtására használható például. A konstruktorhoz **nem** lehet rendelni semmilyen visszatérési értéket, még akkor sem, ha az void lenne.

Egy osztálydefiníció önmagában még semmire sem jó. Hogy használni is tudjuk, először létre kell hoznunk egy osztálypéldányt. Ezt végzi el az utolsó sor:

```
public uj_osztaly = new osztalynev();
```

Ezután az osztály tulajdonságára és metódusára a következő módon tudunk hivatkozni:

```
uj_osztaly.valtozo; uj_osztaly.muvelet();
```

A könyv fejezeteiben bemutatott egyszerű példákban nem fogunk mi magunk explicit létrehozni osztályokat, de tudnunk kell, hogy a színfalak mögött így is számos osztály létrejön. Amikor például Flex komponensekkel dolgozunk és rájuk hivatkozunk, lényegében osztályokat hozunk létre és használunk.

Amennyiben külső fájlokban tárolt osztályokat akarunk használni, azokat importálnunk kell az import utasítással. Például:

```
import air.net.SocketMonitor;
```

3.2. Flex

Az alábbiakban elsajátítjuk azokat az alapismereteket, melyek elengedhetetlenül szükségesek az Apache Flex-szel végzett munkához.

3.2.1. XML forráskódok írása

A Flex forráskód egy XML leírófájl. Az összes látható és nem látható komponenst XML tagek közé beírva kell felvennünk alkalmazásunkba. A Flex komponensei ActionScriptben vannak elkészítve, létrehozásukhoz viszont nem kell egyetlen sor ActionScript kódot sem írnunk. Új projekt létrehozásakor a Flash Builder generál egy kódkeretet.

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"

xmlns:s="library://ns.adobe.com/flex/spark" applicationDPI="160">

<fx:Declarations>
<!-- Place non-visual elements (e.g., services, value objects) here --> </fx:Declarations>

</s:Application>
```

A második sor a Spark névteret definiálja (XMLNS:S). A **Spark** tulajdonképpen a Flex komponenseket tartalmazó architektúra. Létezik egy korábbi architektúra is, az **MX**. Ez utóbbit csak desktop alkalmazásoknál használhatjuk, ott viszont a Spark-kal keverten is. Mobilalkalmazásoknál a Spark komponensek használatára kell törekedni.

Látható, hogy az egész alkalmazás kódja két <s:Application> tag között helyezkedik el, ahol az „:s” szintén a Spark névtér megadását jelenti.

Hasonlóan felépített tag-ek segítségével használhatjuk a többi Flex komponenst is. A sorok tördelése tetszőleges.

A Flex nyelvi lehetőségeivel igen összetett viselkedéssel ruházhatunk fel felhasználói felületeket, ám a valódi alkalmazáslogikát ActionScript-ben kell megírnunk (lásd a 3.3 fejezetet).

3.2.2. Megjegyzések a kódban

XML megjegyzéseket a <!-- … --> határolójelek közé írhatunk. A sorokat a megjegyzéseken belül tetszés szerint tördelhetjük, nincs különbség az egysoros és többsoros megjegyzések írásában. Például:

```
<!--
Ez egy XML megjegyzés
-->
```

3.2.3. Alapvető vizuális Flex komponensek

A négy legalapvetőbb komponens (más néven vezérlő), melyekkel most megismerkedünk, egyszerű szövegek megjelenítését, bevitelét, valamint felhasználói interakciót tesznek lehetővé.

Röviden bemutatjuk a vizuális komponensek felhasználói felületen történő elhelyezésének alapvető módjait is. Az ismertető messze nem teljes és csak olyan szintű bemutatásra vállalkozik, mely a könyv témája szempontjából elengedhetetlen.

3.2.3.1. Pár szóban a Layout-okról

Layout alatt a vizuális komponensek elrendezését értjük a mindenkori felhasználói felületen

A Flex-ben alapvetően háromféle elrendezést használhatunk:

- **Abszolút pozicionálás:** a **BasicLayout** osztály segítségével. Ez az alapértelmezett layout mód is egyben.
- **Relatív pozicionálás:** (a pozicionálás a tartalmazó konténerhez/komponenshez képest történik, ami alapesetben maga az alkalmazás): a **HorizontalLayout, VerticalLayout, TileLayout** osztályok segítségével
- **Constraint-alapú elrendezés:** a komponensek széleinek helyzetét, ill. elhelyezkedését adhatjuk meg a tartalmazó konténerhez/komponenshez képest. A megjelenés tulajdonságait a left, right, top, bottom, horizontalcenter, verticalcenter tulajdonságok beállításával adhatjuk meg. A constraintalapú elrendezés további lehetőségeire ebben a könyvben nem térünk ki. A constraint layout tulajdonképpen a BasicLayout egy speciális felhasználási módja. A constraintalapú elrendezés további lehetőségeire ebben a könyvben nem térünk ki.

A komponensek méretezése szintén háromféleképpen történhet:

- **Alapértelmezett méretezés:** fix, vagy tartalomhoz igazodó automatikus méretezés.
- **Abszolút méretezés:** fix méretek megadása pixelben.

Százalékalapú méretezés. Mindig az adott kijelző, ill. a rendelkezésre álló képernyőfelülethez képest megadott méret, százalékos értékben kifejezve.

Néhány példa layout objektumok használatára.

Horizontális elrendezés

```
<s:layout>
    <s:HorizontalLayout/>
</s:layout>
<s:Button label="Button1"/>
<s:Button label="Button2"/>
<s:Button label="Button3"/>
```

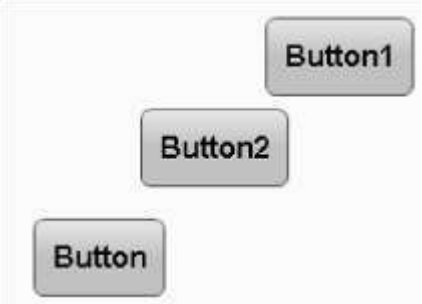
Vertikális elrendezés

```
<s:layout>
```

```
    <s:VerticalLayout/>
</s:layout>
<s:Button label="Button1"/>
<s:Button label="Button2"/>
<s:Button label="Button3"/>
```

Constraint-alapú elrendezés

```
<s:layout>
    <s:BasicLayout/> </s:layout>

<s:Button label="Button1" right="10" top="10"/> <s:Button label="Button2" horizontalCenter="0" top="60"/>
<s:Button label="Button" left="20" top="120"/>
```



3.2.3.2. Általános tulajdonságok

A könyvünkben bemutatott vizuális Flex vezérlőkhöz hozzárendelhetőek az alábbi tulajdonságok. **ID**

Egyedi, szöveges azonosító, ezzel tudunk hivatkozni a komponesre ActionScript és Flex forráskódban. Az elnevezések első karaktere kötelezően betű, a többi karakter lehet speciális karakter, illetve szám is pl.: „Gomb_01”.

X

Komponens vízszintes abszolút koordinátája, pixelben megadva.

Y

A komponens függőleges abszolút koordinátája, pixelben megadva.

A 0,0 helyen levő X és Y koordináták mindig a kijelző bal felső szélét jelentik. A koordinátarendszer ún. jobbsodrású, azaz jobbra és lefelé növekednek az X és Y koordinátaértékek.

Width

A komponens szélessége, pixelben fixen, vagy százalékosan megadva (pl.: 40%).

Height

A komponens magassága, pixelben fixen, vagy százalékban megadva.

Left

A komponens állandó távolsága a tartalmazó konténer (alapesetben a képernyő) bal oldalához képest, pixelben megadva. Felbontásfüggetlen és relatív pozícionáláshoz használhatjuk. **Right**

A komponens állandó távolsága a tartalmazó konténer (alapesetben a képernyő) jobb oldalához képest, pixelben megadva. Felbontásfüggetlen és relatív pozícionáláshoz használhatjuk. **Top**

A komponens állandó távolsága a tartalmazó konténer (alapesetben a képernyő) felső részéhez képest, pixelben megadva. Felbontásfüggetlen és relatív pozícionáláshoz használhatjuk. **Bottom**

A komponens állandó távolsága a tartalmazó konténer (alapesetben a képernyő) alsó részéhez képest, pixelben megadva. Felbontásfüggetlen és relatív pozícionáláshoz használhatjuk.

HorizontalCenter

A komponens helyzete a képernyő, vagy a tartalamzó konténer vízszintes középpontjához képest. Pozitív, vagy negatív, illetve nulla értéket is felvehet. Nulla esetén abszolút vízszintes középre tájolást adhatunk meg.

VerticalCenter

A komponens helyzete a képernyő, vagy a tartalamzó konténer függőleges középpontjához képest. Pozitív, vagy negatív, illetve nulla értéket is felvehet.

FontFamily

A szövegmegjelenítésre használt betűtípus neve.

FontSize

A használt betűtípus mérete.

FontStyle

A használt betűtípus stílusa. Lehetséges értékei: normal, italic.

FontWeight

A megjelenített szöveg vastagsága. Lehetséges értékei: normal, bold. **LineThrough**

A megjelenített szöveg áthúzott megjelenítése. Lehetséges értékei: true, false.

TextDecoration

A megjelenített szöveg aláhúzott megjelenítése. Lehetséges értékei: none, underline.

TextAlign

A megjelenített szöveg vízszintes igazítása. Lehetséges értékei: center, end, justify, left, right, start.

VerticalAlign

A megjelenített szöveg függőleges igazítása.

Lehetséges értékei: bottom, justify, middle, top.

Természetesen nem kötelező mindig az összes tulajdonságot megadnunk. Egyes tulajdonságok egymás ellen is hathatnak, pl. a Left és az X tulajdonságok egyidejű használata ellenjavallott.

Példa egy erősen „túldekorált” Label komponensre:

```
<s:Label id="Label1" x="10" y="20" width="217" height="46" fontFamily="Times New Roman" fontSize="17" fontStyle="italic" fontWeight="normal" lineThrough="true" text="hello" textAlign="center" textDecoration="underline" verticalAlign="middle"/>
```

Visible

A komponens láthatóságát befolyásolhatjuk. Lehetséges értékei: true, false.

3.2.3.3. A Label komponens

A Label komponens statikus, azaz közvetlenül nem szerkeszthető szövegek megjelenítésére használható.

Text

A szövegkomponens által megjelenített szöveg.

Példa:

```
<s:Label id="Label1" x="10" y="20" text="hello"/>
```

3.2.3.4. A Button komponens

Nyomógomb, felhasználói interakció fogadására (valamilyen parancs kiadása, funkció aktiválása, stb.) használható.

Label

A nyomógomb felirata. **Enabled**

Aktívvá, vagy inaktívvá tehetjük a nyomógombot. Inaktív állapotban kiszürkítve jelenik meg. Lehetséges értékei: true, false. **Icon**

Egyedi képet jeleníthetünk meg a nyomógombon. A következő képformátumok támogatottak: PNG, JPG, GIF, SWF, FXG.

Szabályozhatjuk, hogy a képfájlok az alkalmazásba ágyazódjanak be, vagy különálló fájlként települjenek a kész alkalmazással. A két megoldás megadási módjára példák:

```
icon="@Embed('assets/kep.png')" icon="assets/kep.png"
```

3.2.3.5. A TextInput komponens

Egysoros szövegbevitelre alkalmas vezérlő.

Text

A beviteli mező tartalma.

Prompt

Ezzel a tulajdonsággal megadhatunk egy szöveget, ami akkor jelenik meg, ha a beviteli mező teljesen üres. **maxChars**

A beviteli mezőbe írható karakterek maximális számát korlátozhatjuk.

Enabled

Aktívvá, vagy inaktívvá tehetjük a komponenst. Inaktív állapotban kiszürkítve jelenik meg és nem lehetséges a szövegbevitel. Lehetséges értékei: true, false.

3.2.3.6. A TextArea komponens

Többsoros szövegbevitelre alkalmas komponens.

Text

A beviteli mező tartalma.

Prompt

Ezzel a tulajdonsággal megadhatunk egy szöveget, ami akkor jelenik meg, ha a beviteli mező üres.

maxChars

A beviteli mezőbe írható karakterek maximális számát korlátozhatjuk.

Editable

A megjelenített szöveg módosíthatóságát szabályozhatjuk. Nem összekeverendő az Enabled tulajdonsággal! Lehetséges értékei: true, false.

Enabled

Aktívvá, vagy inaktívvá tehetjük a komponenst. Inaktív állapotban kiszürkítve jelenik meg és nem lehetséges a szövegbevitel. Lehetséges értékei: true, false.

3.3. Az ActionScript kód beágyazása Flex kódba

Flex alkalmazásunk forráskódjában eddig csak XML tag-eket adtunk meg, most megnézzük, hogyan kell elhelyezni ActionScript nyelven írt forráskódot Flex alkalmazás forráskódjában.

Actionscript kódot a Flex névtérben megadott Script tag között kell megadnunk, egy speciális <![CDATA[…]]> tag közé. A Flex innen „fogja tudni”, hogy a CDATA-n belül megadott szöveg ActionScript forráskód.

```
<fx:Script> <![CDATA[

// ActionScript kód helye
```

```
]]>
</fx:Script>
```

4. Példaprogramok

Ebben és a következő fejezetben konkrét példákat, komplett kisalkalmazásokat fogunk áttekinteni.

Az alkalmazáskészítés alapfogásainak megismerése során egyre bonyolultabb alkalmazásokat fogunk készíteni, míg végül eljutunk az adatbázis-kezeléshez.
Első lépésként a modern alkalmazásfejlesztés egyik alappillérét ismerjük meg, az eseménykezelést.

4.1. Inicializációs függvény írása

Eseménykezelés alatt – egyszerűen megfogalmazva - azt értjük, hogy egy program egy, a felhasználó, az operációs rendszer vagy saját maga által kiváltott esemény bekövetkeztekor valamit csinál. **Esemény** lehet egy nyomógomb lenyomása, egy hiba bekövetkezte stb.

Gyakori, hogy alkalmazásuk indításakor automatikusan inicializálni kell értékeket, beállításokat, ellenőrzéseket kell elvégezni. Erre szolgál az alkalmazás applicationComplete nevű eseménye, mely akkor váltódik ki, ha az alkalmazás minden belső objektumot létrehozott, pl. a felhasználói felület elemeit. Az eseményhez egy ún. **eseménykezelő függvényt** fogunk készíteni.

Hozzunk létre egy új mobilprojektet 'Pelda2' néven!

Szükségünk lesz egy TextInput komponensre, melynek neve (ID-ja) legyen 'beviteli_mezo', a mező Text tulajdonsága pedig a „Szöveg” szöveget tartalmazza!

A Flex forráskódban az Application tag-en belülre írjuk be: 'application' majd nyomjuk le a CTRL+SPACE billentyűkombinációt és válaszszuk ki az applicationComplete listaelemet a felkínált lehetőségek közül. A Flash Builder ekkor felajánlja, hogy elkészíti számunkra az eseménykezelő metódust ('Generate ApplicationComplete Handler'). Nyomjunk erre is ENTER-t. Megjelenik a generált metódus neve, amit rögtön írjunk is át 'init'-re!

A Flash Builder automatikusan létrehozta a metódus kódkeretét is egy ActionScript kódblokkban. Ezenkívül Actionscript kódunk elejéhez hozzáadta az **import** mx.events.FlexEvent; sort, mivel az eseménykezeléshez a FlexEvent osztályra van szükség.

Természetesen kézzel is beírhattuk volna mindezeket, de a Content Assist szolgáltatás segítségével jóval gyorsabban haladhatunk. Metódusunk fejléce:

```
public function init(event:FlexEvent):void
```

A metódusnak átadott eseményosztály példánya ('event') különböző értékeket kap az esemény bekövetkeztekor, melyeket feldolgozhatunk a metódusban (példánkban erre nem lesz szükség). A fenti lépésekkel elértük, hogy alkalmazásunk meghívja, azaz lefuttassa az 'init' metódust, amikor az elindul.
Hozzunk létre egy szöveges és egy decimális egész típusú változót a metódusunk törzsében!

```
var szoveg:String;
```

```
var szam:int;
```

Ezután adjunk értéket a két változónak!

```
szam=5;
szoveg="Kedvenc osztályzatom: " + szam.toString(10);
```

A második sor némi magyarázatra szorul. A szam változót átalakítjuk szöveges adattá és ezt hozzáfűzzük a szoveg változó tartalmához. A konverzióhoz tulajdonképpen a szam objektum egyik metódusát hívtuk meg (.toString), ami sztringgé alakítja a tárolt számértéket (azaz az 5-öt). (Ne feledjük, hogy a String adattípus maga is osztály, mely rendelkezik metódusokkal!)

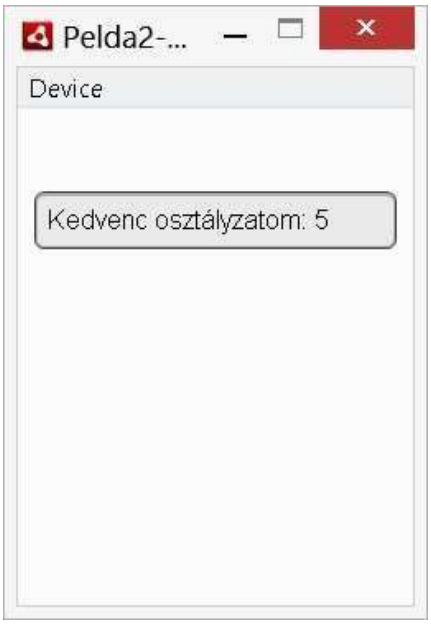
Utolsó lépésként a szoveg változónk tartalmát másoljuk a beviteli mezőbe:

```
beviteli_mezo.text=szoveg;
```

Példaprogramunk teljes forráskódja:

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
xmlns:s="library://ns.adobe.com/flex/spark" applicationDPI="160"
applicationComplete="init(event)">

<fx:Script> <![CDATA[
import mx.events.FlexEvent;
public function init(event:FlexEvent):void
{
var szoveg:String; var szam:int;

szam=5;
szoveg="Kedvenc osztályzatom: " +  szam.toString(10); beviteli_mezo.text=szoveg;

}
]]>
</fx:Script>

<s:TextInput id="beviteli_mezo" left="10" right="10" top="50" width="100%" text=""/>

</s:Application>
```

Próbáljuk ki alkalmazásunkat, futtassuk az alkalmazást szimulált módban (vagy mobileszközön, ha szeretnénk)! Ha mindent jól csináltuk, az alábbihoz hasonló eredményt kell kapnunk:



4.2. Gomblenyomás kezelése

A nyomógombok fontos részei a felhasználói felületeknek, ezért alapszintű kezelésükkel tisztában kell lennünk. Hozzunk létre egy új, 'Pelda3' nevű projektet! A felhasználói felülethez először is hozzáadunk egy szövegbeviteli mezőt:

```
<s:TextInput id="beviteli_mezo" left="10" right="10" top="30" text=""/>
```

Majd két nyomógombot:

```
<s:Button id="Gomb_Csokkent" left="10" top="100" width="80%" label="Csökkent"/>

<s:Button id="Gomb_Novel" left="10" top="150" width="80%" label="Hozzáad"/>
```

A gombok lenyomásához funkciókat rendelünk, ehhez külön függvényeket fogunk írni, melyek frissíteni fogják a szövegmező tartalmát.

Alkalmazásunkkal egy egész szám értékét növelhetjük/csökkenthetjük eggyel, 1-10-ig terjedő értékhatárok között. A szám értékét pedig meg is jelenítjük.

Az ehhez szükséges kódot ActionScript-ben írjuk meg. Az alkalmazáslogikát a 'Kivon' és 'Hozzaad' nevű függvények fogják tartalmazni:

```
protected function Kivon(event:MouseEvent):void
{
        if (szam == 1) return;
        --szam;
        szoveg="Szám értéke: "+szam.toString(10);
}                                                    beviteli_mezo.text=szoveg;
```

```
protected function Hozzaad(event:MouseEvent):void
{
 if (szam == 10) return;
 ++szam;
 szoveg="Szám értéke: "+szam.toString(10);  beviteli_mezo.text=szoveg;
}
```

A Kivon függvény először is a szam változó értékét vizsgálja meg. Amennyiben ez 1, kilépünk a függvényből (return; utasítás). Ellenkező esetben a szám értékét csökkentjük eggyel és a már megismert módon frissítjük a szöveges mezőt.

A Hozzaad függvény is ugyanígy működik, csak éppen a felső értékhatár vizsgálatát és a szám értékének növelését tartalmazza.

Egy apró egyszerűsítést még el kell végeznünk a forráskód szerkezetén. A szoveg és a szam változóink deklarációját helyezzük az init függvényen kívülre, így „globális" Vátozókként használhatjuk őket. Az egyértelműség kedvéért a hozzáférési módként public-ot adjunk meg, így minden függvény/metódus számára elérhetőek lesznek.

```
public var szoveg:String; public var szam:int;
```

Utolsó lépésként pedig a függvényeket hozzárendeljük a nyomógombok „kattintás" eseményéhez:

```
<s:Button id="Gomb_Csokkent" left="10" top="100" width="80%" label="Csökkent"
click="Kivon(event)"/>

<s:Button id="Gomb_Novel" left="10" top="150" width="80%" label="Hozzáad"
click="Hozzaad(event)"/>
```

Programunk teljes forráskódja:

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application  xmlns:fx="http://ns.adobe.com/mxml/2009"
xmlns:s="library://ns.adobe.com/flex/spark"
applicationDPI="160"
applicationComplete="init(event)">

<fx:Script>  <![CDATA[
import mx.events.FlexEvent;

public var szoveg:String; public var szam:int;

protected function init(event:FlexEvent):void
{
 szam=5;
 szoveg="A szám értéke: "+szam.toString(10);  beviteli_mezo.text=szoveg;
}

protected function Kivon(event:MouseEvent):void
{
 if (szam == 1) return;
 --szam;
 szoveg="A szám értéke: " +szam.toString(10);  beviteli_mezo.text=szoveg;
}

protected function Hozzaad(event:MouseEvent):void
{
 if (szam == 10) return;
 ++szam;
 szoveg="A szám értéke: " +szam.toString(10);  beviteli_mezo.text=szoveg;
}

]]>
</fx:Script>

<s:TextInput id="beviteli_mezo" left="10" right="10" top="30" text=""/>

<s:Button id="Gomb_Csokkent" left="10" top="100" width="80%" label="Csökkent"
click="Kivon(event)"/>

<s:Button id="Gomb_Novel" left="10" top="150" width="80%" label="Hozzáad"
click="Hozzaad(event)"/>

</s:Application>
```

Végezzünk el egy tesztfuttatást!

4.3. Input-output események kezelése

Akármilyen külső erőforráshoz akarunk hozzáférni, különösen, ha hálózati erőforrásról van szó, nagyon fontos, hogy alkalmazásaink fel legyenek készítve arra az eshetőségre, ha az erőforráshoz, például egy szöveges fájlhoz, nem tudnak hozzáférni.

Mi a hibakezelésnek egy egyszerű, de hatékony formájával fogunk megismerkedni. Az input-output (kivitel-bevitel) műveletek sikertelensége (és sikeressége is) ugyanis egy eseményt generál, aminek a kezelését alkalmazásszinten át tudjuk venni. A következő, 'Pelda4' nevű példaalkalmazás erre fog egy példát bemutatni. A program segítségével tetszőleges szövegalapú erőforrást tölthetünk be és jeleníthetünk meg. A tetszőleges itt tényleg szó szerint értendő: akár HTML weblapok elérhetőségét is megadhatjuk, ekkor a HTML forráskód fog megjelenni a programban.

Az alkalmazás szöveges alapú hálózati erőforrásokat fog betölteni és megjeleníteni egy szövegmezőben. Kommunikációs hiba esetén hibaüzenetet jelenít meg. Az erőforrások URLjét egy szöveges beviteli mezőben adhatjuk meg.

A hálózati kommunikációhoz két ActionScript osztályt fogunk használni, melyek alapesetben erőforrások betöltését teszik lehetővé:

URLRequest: ezzel az osztállyal adjuk meg az erőforrás helyét.

URLLoader: ez az osztály végzi el az erőforrás betöltését.

Az olvas_Loader nevű URLLoader osztálypéldányt elég egyszer létrehozni, a program legelején.

public var olvas_Loader:URLLoader = **new** URLLoader();

Az alkalmazás külön hibakezelő függvénnyel kezelni fogja a speciális, IOErrorEvent.IO_ERROR eseményt.

Az erőforrás sikeres betöltésének végén szintén egy esemény, az Event.COMPLETE fog kiváltódni, ami a betöltési művelet sikeres befejeztét jelzi. Csak ezután szabad megkísérelni a betöltött erőforrás feldolgozását.

Mivel ActionScriptből közvetlenül akarunk eseményeket kezelni, ezért mindenképpen importálnunk kell a FlexEvent osztályt: **import** mx.events.FlexEvent;

A betöltés végének kezelése aszinkron módon kell, hogy történjen. Ezért valójában két függvényre lesz szükségünk. Egyre a betöltés kezdeményezéséhez és egy másodikra a betöltött adat feldolgozásához, ami esetünkben egyszerű megjelenítést jelent majd. A második függvény csak akkor fog végrehajtódni, ha az első függvényben elindított betöltés sikeresen befejeződött.

Két függvényünk neve ez lesz:

- adat_olvas_kezdes
- adat_olvas_befejezes

A függvények kódja:

```
public function adat_olvas_kezdes():void
{
//URL átvétele a beviteli mezőből  var olvas_URL:URLRequest= new URLRequest(beviteli_mezo.text);
//az URL betöltésének indítása
olvas_Loader.load(olvas_URL);
}
public function adat_olvas_befejezes(e:Event):void
{
//a betöltött erőforrás megjelenítése  kimenet.text=e.target.data;
}
```

Az erőforrás tartalmát az adat_olvas_befejezes függvény egy Event objektumon keresztül kapja meg. Az objektum target.data tulajdonsága tartalmazza magát az adatot (lásd fenti kódrészletet).

A felhasználói felületen egy egysoros szövegbeviteli mezőre, alatta két gombra, majd egy nagyobb méretű, többsoros szöveg megjelenítésére alkalmas szövegdobozra lesz szükségünk.

```
<s:TextInput id="beviteli_mezo" left="10" right="10" top="10" horizontalCenter="0" prompt="Írjon be egy adatot!" text="" textAlign="left"/>

<s:TextArea id="kimenet" left="10" right="10" top="160" bottom="10" height="100%" horizontalCenter="0"/>

<s:Button id="Gomb_olvasas" left="10" right="10" top="50" label="Olvasás" horizontalCenter="0"/>

<s:Button id="Gomb_Kilep" left="10" right="10" top="100" label="Kilépés" horizontalCenter="0"/>
```

ActionScriptben eseményeket objektumokhoz az alábbi két metódussal tudunk rendelni ill. eltávolítani:

addEventListener(esemény, eseménykezelő_függvény) removeEventListener(esemény, eseménykezelő_függvény)

Alkalmazásunk indításakor egy inicializációs függvényt fogunk lefuttatni, mely elvégzi az eseménykezelők hozzárendelését az URLLoader objektumpéldányokhoz:

```
protected function init():void
{
olvas_Loader.addEventListener( IOErrorEvent.IO_ERROR, IO_hiba);

olvas_Loader.addEventListener( Event.COMPLETE,adat_olvas_befejezes);
}
```

Amennyiben az erőforrás nem elérhető (pl. nincsen internetkapcsolat, vagy egyszerűen rossz linket adunk meg), akkor egy IO_ERROR esemény generálódik. Ha ezt nem kezeljük, akkor alkalmazásunk mobileszközön futtatva jó eséllyel le fog állni, PC-n szimulálva pedig egy hibaüzenet fog megjelenni futtatáskor. Ezt a kellemetlenséget elkerülhetjük már azzal is, hogy egy megfelelő függvénnyel kezeljük ezt a hibaeseményt. Ez a függvény az IO_hiba lesz és egyszerűen egy üzenet ír ki a szövegdobozba, ha az erőforrás betöltése nem sikerült.

```
public function IO_hiba(event:IOErrorEvent):void
{
 kimenet.text="Hálózati kommunikációs hiba.";
}
```

Már csak egy dolog van hátra: az alkalmazás rendeltetésszerű bezárásáért felelős függvény kódjának megírása. Az androidos alkalmazások ugyanis alapesetben nem állnak le a mobileszköz „Vissza”, vagy „Home” gombjának lenyomásakor, csupán a háttérbe kerülnek. Ezért a teljes leállítást mindig a NativeApplication.nativeApplication.exit(); utasítással kell elvégeznünk. A kilépés során eltávolítjuk az eseménykezelőket is:

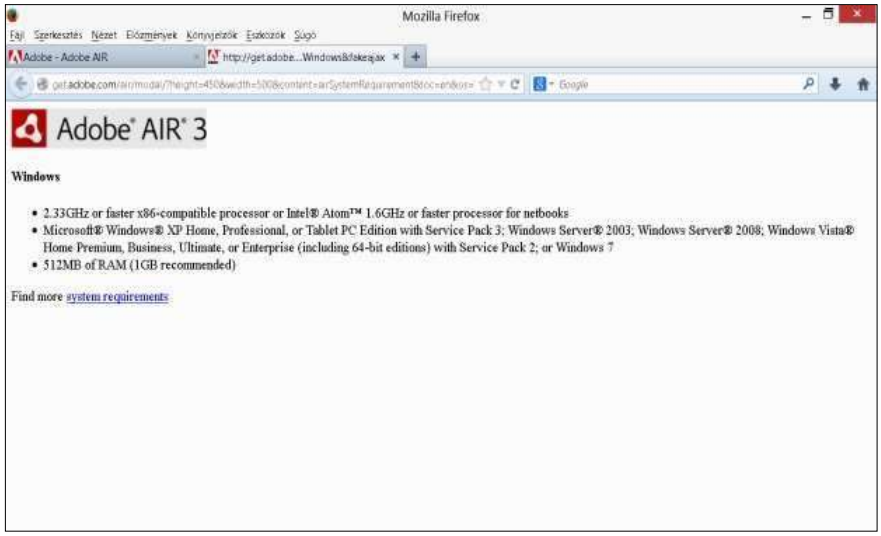
```
protected function Kilepes():void
{
olvas_Loader.removeEventListener( Event.COMPLETE,adat_olvas_befejezes);

olvas_Loader.removeEventListener(
IOErrorEvent.IO_ERROR, IO_hiba);
      NativeApplication.nativeApplication.exit();
}
```

A teljes alkalmazás forráskódja:

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application  xmlns:fx="http://ns.adobe.com/mxml/2009"
xmlns:s="library://ns.adobe.com/flex/spark"
applicationDPI="160" applicationComplete="init()">

<fx:Script> <![CDATA[  import
mx.events.FlexEvent;

public var olvas_Loader:URLLoader = new URLLoader();

protected function init():void
{
olvas_Loader.addEventListener(IOErrorEvent.IO_ERROR, IO_hiba); olvas_Loader.addEventListener(
Event.COMPLETE,adat_olvas_befejezes);
}

protected function Kilepes():void
{
olvas_Loader.removeEventListener( Event.COMPLETE,adat_olvas_befejezes);

olvas_Loader.removeEventListener(
IOErrorEvent.IO_ERROR, IO_hiba);
NativeApplication.nativeApplication.exit();
}

public function IO_hiba(event:IOErrorEvent):void
{
kimenet.text="Hálózati kommunikációs hiba.";
}

public function adat_olvas_kezdes():void
{
```

```
//URL átvétele a beviteli mezőből var olvas_URL:URLRequest=new URLRequest(beviteli_mezo.text);
//az URL betöltésének indítása olvas_Loader.load(olvas_URL);
}

public function adat_olvas_befejezes(e:Event):void
{
//a betöltött erőforrás megjelenítés  kimenet.text=e.target.data;
}
]]>
</fx:Script>

<s:TextInput id="beviteli_mezo" left="10" right="10" top="10" horizontalCenter="0" prompt="Írjon
be egy
adatot!" text="" textAlign="left"/>

<s:TextArea id="kimenet" left="10" right="10" top="160" bottom="10" height="100%"
horizontalCenter="0"/>

<s:Button id="Gomb_olvasas" left="10" right="10" top="50" label="Olvasás" horizontalCenter="0"
click="adat_olvas_kezdes()"/>

<s:Button id="Gomb_Kilep" left="10" right="10" top="100" label="Kilépés" horizontalCenter="0"
click="Kilepes()"/>

</s:Application>
```

Most pedig futtassuk alkalmazásunkat! Az alábbi képen az Adobe AIR rendszerkövetelményeit leíró oldal URL-jének tartalma látható egy hagyományos asztali böngészőprogramban.



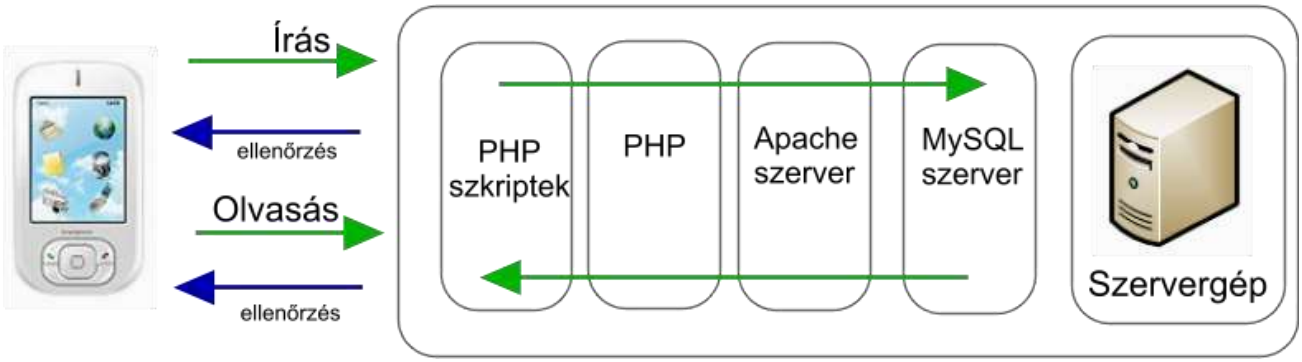
Ugyanezt az URL-t megadva alkalmazásunknak, az betölti az oldal forráskódját:


Pelda4-debug
Device
http://get.adobe.com/air/modal/?height=450&width=500&conten
Olvasás
Kilépés
<style>.ui-dialog { background:#eeeeee;}</style><h1></h1><h4 id="windows">Windows</h4>

5. Adatbázis-kezelő mobilalkalmazás készítése

Ez a fejezet egy demonstrációs célt szolgáló, de működő adatbázis-kezelő alkalmazás elkészítését mutatja be.

Az alábbi ábra tervezett alkalmazásunk kommunikációs modelljét szemlélteti. Alapvetően egy írási és egy olvasási műveletet kell megvalósítanunk.



A kihívás természetesen az, hogy a mobilalkalmazásunk egészen a szerver MySQL szintjével képes legyen kommunikálni, a többi rétegen keresztülhaladva és adatot is tudjon fogadni onnan.

Alkalmazásunk egy DEMO_TABLA nevű adattáblát fog tudni kezelni a 'test' nevű adatbázisban.

Az adattábla egyetlen adatmezővel fog rendelkezni, melynek neve 'adatmezo' és rövid szövegsorok tárolására lesz alkalmas.

Az adatbázist létrehozó SQL lekérdezés:

```
CREATE  TABLE `test`.`DEMO_TABLA`
(`adatmezo` VARCHAR(50) NOT NULL);
```

5.1. Adatcsere PHP szkriptekkel

A PHP szkriptek futtatását ActionScript szintjén fogjuk kezdeményezni és ott is dolgozzuk fel a visszakapott eredményeket. Az adatbázissal közvetlenül két PHP szkript fog kommunikálni, az egyik olvasni fog az adatbázisból, a másik beírni fog az adatbázisba.

A mobilalkalmazásunk PHP szkripteknek fog tudni adatot átadni és értékeket fogadni azok kimenetéről. ActionScript-en keresztül elérhető a PHP szkriptek kimenete is (a print utasítások a php kódjában). Ez lehetővé teszi a műveletek sikerességének az ellenőrzését, ill. az adatbázisból kiolvasott adatok lekérését.

Ahhoz, hogy befolyásunk legyen a PHP szkriptek által átadott adatok tartalmára, először is azt kell tisztáznunk, hogy PHP szkripteknek hogyan lehet adatot átadni. Erre két módszer van: ún. **GET** és **POST** átadási mód. Mi ez utóbbit fogjuk használni, de szót ejtünk a GET-ről is, hogy átfogóbb képet kapjunk.

Hozzunk létre egy 'teszt_PHP_GET.php' szkriptet, melynek tartalma ez:

```
<?php
$adat = $_GET["adat"];
print(”$adat”); print("<BR>");  print("Rendben lefuttatva.");
?>
```

A szkriptet böngészőben az alábbi sorral tudjuk lefuttatni:

```
http://localhost/teszt/teszt_PHP_GET.php?adat= tesztszöveg
```

A böngésző a következő szöveget jeleníti meg:

tesztszöveg
Rendben lefuttatva.

GET módszerrel tehát egy „?” jellel jelezzük, hogy adatot adunk át a szkriptnek, utána a szkriptben használt PHP változó nevét, majd tartalmát kell megadnunk. Egyszerre több adatot is átadhatunk, ilyenkor egy „&” jelet kell az adatok közé írnunk. Például:

```
localhost/teszt/szkript.php?adat1=szöveg1&adat2= szoveg2
```

A GET egyik legnagyobb hátránya, hogy korlátozott mennyiségű adatot lehet vele elküldeni. Ez szerveroldali beállítástól is függ, de általában kb. 2000 bájtnál nem lehet több.

A POST típusú adatátadásnál ilyen megkötések nincsenek és szerencsére a fenti szkriptet sem kell teljesen átírni. Lényegében csak a 'GET' szót kell 'POST'-ra kicserélni a szkript szövegében (pl. 'teszt_PHP_POST.php').

```
<?php
$adat = $_POST["adat"]; print(”$adat”); print("<BR>");  print("Rendben lefuttatva.");
?>
```

A POST paraméterek átadásáról a hívó alkalmazásnak/weboldalnak kell alkalmazáslogika szintjén gondoskodnia. Ennek a technikáját fogjuk most megismerni.

5.2. Adatbázis-kezelő PHP szkriptek megírása

Négyféle adatra van szükség a szkriptekhez: a host nevére (ami tesztkörnyezetben 'localhost', éles alkalmazás esetén a megfelelő **szervernév,** ill. külső szolgáltató esetén a szolgáltató adja meg a nevet), egy **felhasználói név** és **jelszó** páros az adatbázishoz/adattáblához, valamint a bemenő **adat** (ez csak az adatbázisba történő íráshoz kell).

Az első szkriptünk neve 'olvas.php', ezzel fogunk olvasni az adatbázisból.

```
<?php
$hostname='localhost';
$nev='nev';
$jelszo='XYZ';

if (!$kapcsolat = @mysql_connect($hostname, $nev, $jelszo)) {die(print("Adatbázishiba: nincs kapcsolat."));} mysql_set_charset('utf8',$kapcsolat);

if (!mysql_select_db('test',$kapcsolat)) {die(print("Adatbázishiba: nincs adattábla."));}

$parancs = "select * from demo_tabla";

if(!$eredmeny = mysql_query($parancs, $kapcsolat)) {die(print("Adatbázishiba: lekérdezési hiba."));}

while($row=mysql_fetch_array($eredmeny, MYSQL_ASSOC))
{
 $adatmezo=$row['adatmezo'];
 print("$adatmezo"); print("<BR>");
}

mysql_close($kapcsolat); ?>
```

A szkript először létrehozza az adatbáziskapcsolatot, majd az 'UTF-8' karakterkódolást állítja be, ezzel biztosítva a megfelelő karaktermegjelenítést.

Második lépésként a szkript a DEMO_TABLA adattáblát választja ki munkatáblának.

Egy $parancs nevű változóban tároljuk a lekérdezést, amit a mysql_query függvénnyel hajtunk végre.

A szkript ezután egy ciklus segítségével elkezdi folyamatosan kiolvasni és kiírni a sorokat az adattáblából.

Utolsó lépésként az adatbázis lezárására is sor kerül.

Bármilyen hiba esetén a szkript hibaüzenetet ír ki és befejezi működését.

Most pedig lássuk az adattáblába történő beírás szkriptjét, melynek neve 'kiir.php':

```
<?php
$hostname='localhost';
$nev='nev';
$jelszo='XYZ';
$adat = $_POST["adat"];

if (!$kapcsolat = @mysql_connect($hostname, $nev, $jelszo)) {die(print("Adatbázishiba: nincs
kapcsolat."));} mysql_set_charset('utf8',$kapcsolat);

if (!mysql_select_db("test",$kapcsolat)) {die(print("Adatbázishiba: nincs adattábla."));}

$parancs = "INSERT INTO demo_tabla VALUES
('".$adat."')";
if(!$eredmeny = mysql_query($parancs, $kapcsolat)) {die(print("Adatbázishiba: lekérdezési
hiba."));}

mysql_close($kapcsolat);  print("Sikeres adatkiírás!");
?>
```

A szkriptben megtörténik az elküldött bemenő adat feldolgozása is, erre az $adat nevű változót használjuk.

Megjegyzés:
Régebbi MySQL verzióknál előfordulhat, hogy nem támogatják a mysql_set_charset függvényt.

5.3. PHP szkriptek kezelése ActionScript-ből

PHP szkriptjeink futtatásához az előző fejezetben megismert alábbi két osztályt fogjuk használni:

URLRequest: ezzel az osztállyal adjuk meg a szkriptek URL-jét és az átadni kívánt adatokat.

URLLoader: ez az osztály végzi el a PHP szkriptek lefuttatását.

Az implementálás során három nehézséggel kell megbirkóznunk:

- **Szkriptfuttatás ellenőrizhetősége - Aszinkron feldolgozás biztosítása - Karakterkódolás**

5.3.1. Szkriptfuttatás ellenőrizhetősége

Felmerül a kérdés, hogy az ActionScript program honnan fogja tudni, hogy végrehajtódotte a PHP szkript? Ezt eseménykezelésen keresztül lehet megvalósítani. Legalább háromféle eseményre fel kell készítenünk programunkat:

- a PHP szkript futtatása sikeres volt - a
futtatás során valamilyen hiba lépett fel - a
PHP szkript fizikailag nem volt elérhető.

5.3.2. Aszinkron feldolgozás biztosítása

A dolgokat tovább bonyolítja, hogy egy távoli szkript elérése, lefutása, majd a futás eredményének visszajelzése akár több másodpercet is igénybe vehet. Ezalatt az idő alatt programunk futása már régen „máshol" járhat. A szkriptek kezelését ezért **aszinkron** módon kell megvalósítani. Szerencsére ennek a mikéntjét már megismertük, ezt figyelembe véve készítettük el a 'Pelda4' alkalmazásunkat is.

5.3.3. Karakterkódolás

A helyes karakterkódolás alapvető fontosságú az adatcsere folyamán. A leggyakoribb, mármár klasszikusnak nevezhető probléma az ékezetes karakterek megjelenítésénél merül fel. Szerveroldalon, PHP szkriptjeinkkel hiába állítunk be UTF-8 kódolást az adatbázisban tárolt adatok kezeléséhez, számos más szerver- és kliensoldali beállítás következtében az eredmény még ekkor is eltérhet a reméltől.

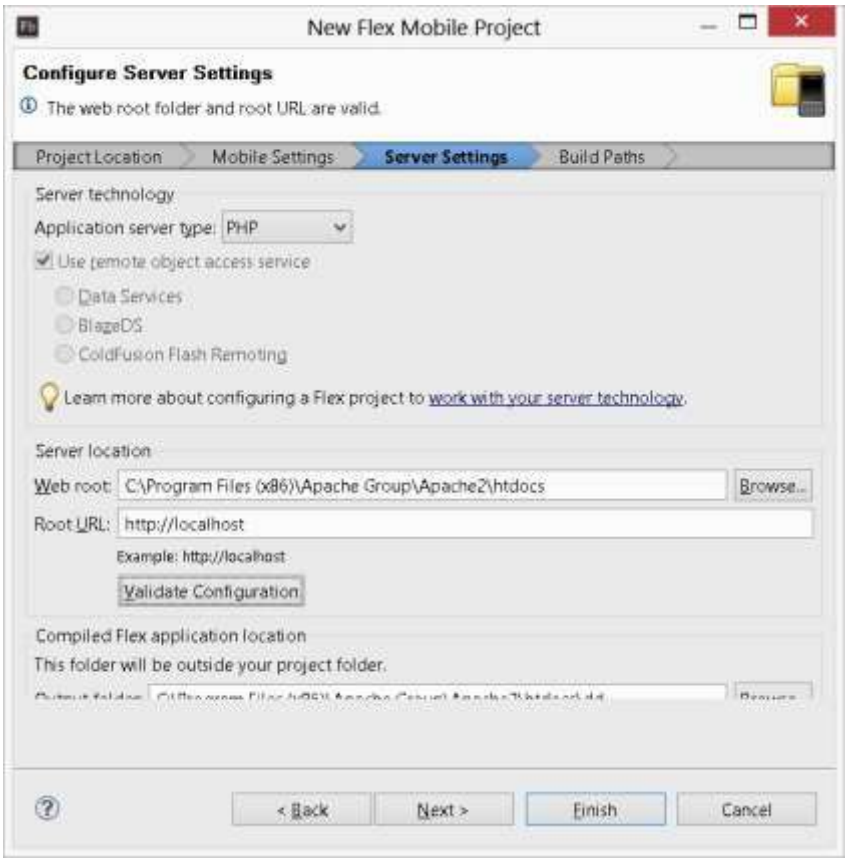
Helyes módszer lehet, ha az ékezetes karaktereket eleve HTML kódokká alakítva tároljuk az adatbázisban és kiolvasáskor visszaalakítjuk azokat. Erre a PHP is lehetőséget biztosít.

Az ékezetes karakterek dilemmája mellett keresztplatformos megoldásoknál az ún. újsor karakterek tárolásának problémája is felmerülhet, a UNIX és a Windows ugyanis nem ugyanúgy tárolja az új sort jelző karaktert.

A szerveroldalon alkalmazott http szerver továbbá küldhet járulékos adatokat (pl. jelzőkarakterek formájában) a PHP szkriptek kimenete elé biggyesztve. Ezekre a kódolási nehézségekre itt most csak utalunk, mindössze azt kívánjuk tudatosítani, hogy mindig nagyon alaposan teszteljük a kommunikációs rendszer minden egyes rétegét!

Hozzunk létre a Flash Builderben egy új, 'Pelda5' nevű projektet! Az alkalmazás elkészítése előtt be kell állítanunk az alkalmazott szervertechnológiát. Ezt a későbbiekben is bármikor megtehetjük a Project → Properties → Flex Server menüben.

Ügyeljünk arra, hogy a Flash Builder-t ebben az esetben rendszergazdai jogosultságokkal futtassuk, mivel rendszermappához állítunk be hozzáférést. Máskülönben a Flash Builder meg fogja tagadni a beállítás elvégzését.



Az 'Application server type' opciónál válaszszuk ki a PHP-t!

A 'Web root' és 'Root URL' megadása szintén kötelező. Itt a webszerverünk elérhetőségét kell megadnunk.

Lokális szerverhasználat (pl. fejlesztés alatt, teszteléskor) esetén a 'Web root' értéke ehhez hasonló lesz:

C:\Program Files (x86)\Apache Group\Apache2\htdocs A 'Root URL' pedig: http://localhost

Végezetül a 'Validate configuration' gombra kattintva ellenőriztetnünk kell a szerver elérhetőségét. A Flash Builder csak sikeres érvényesítés után hajlandó elmenteni a beállításokat. Az Apache szervernek feltétlenül futnia kell.

Megjegyzés: Az éles alkalmazás publikálásakor / tesztelésekor a FlashBuilder kérheti a szerver elérhetőségének módosítását a lokális adatokról (amennyiben lokálisan teszteltük alkalmazásunkat) a hálózati adatokra. A tapasztalat azt mutatja, hogy ennek a beállításnak az elvégzése nem kötelező, az éles alkalmazás ugyanolyan jól működik a fenti lokális beállításokkal is.

Alkalmazásunk felületén két szöveges beviteli mezőre és három nyomógombra lesz szükségünk, az alábbi elrendezésben.

A legfelső komponens egy TextInput komponens lesz, ide írhatjuk majd be az adatbázisba felvinni kívánt adatokat. Itt egysoros szövegeket tudunk megadni. A komponens ID-ja legyen 'beviteli_mezo' !

Az 'Kiírás' feliratú nyomógomb segítségével küldhetjük el az adatokat az adatbázisba, míg az 'Olvasás' feliratú gombbal az adattáblánk tartalmát (az összes adatsort) beolvashatjuk az alkalmazásunkba, amit egy TextArea komponenssel fogunk megjeleníteni, melynek ID-ja 'Kimenet' legyen! Ezt a PHP szkript kimenetének megjelenítésére fogjuk használni.

A 'Kilépés' feliratú nyomógombot az alkalmazás rendeltetésszerű bezárására használhatjuk.

Próbáljuk meg egyedül beállítani a komponensek left, right, top, bottom, width és height tulajdonságait úgy, hogy különböző felbontásokkal és tájolással is megfelelően és ízlésesen jelenjenek meg!

Arra az esetre, ha valahol elakadna az Olvasó, az alábbi kódrészletekből lehet puskázni:

```
<s:TextInput id="beviteli_mezo" top="8" left="10" right="10" horizontalCenter="0" prompt="Írjon be egy adatot!" text="" textAlign="left"/>

<s:Button id="Gomb_kiiras" top="48" left="10" right="10" label="Kiírás" click="adat_kiir_kezdes()" horizontalCenter="0"/>

<s:TextArea id="kimenet" left="10" right="10" top="195" bottom="10" horizontalCenter="0"/>

<s:Button id="Gomb_olvasas" top="96" left="10" right="10" label="Olvasás" horizontalCenter="0" click="adat_olvas_kezdes()"/>

<s:Button id="Gomb_Kilep" top="144" left="10" right="10" label="Kilépés" horizontalCenter="0" click="Kilepes()"/>
```

A rend kedvéért adhatunk a nyomógomboknak is ID-t, de példánkban ez akár el is hagyható, mivel nem hivatkozunk rájuk ActionScriptből.

Mivel alkalmazásunk eseményeket is kezel, ezért nagyon fontos, hogy forráskódunk legelején importáljuk a megfelelő osztályokat, amenynyiben a FlashBuilder ezt nem tette meg automatikusan.

```
import mx.events.FlexEvent;
```

Szükségünk lesz egy speciális 'adatok' nevű objektumra, ennek segítségével fogjuk átadni az adatokat a PHP szkripteknek. Ez egy speciális, URLVariables típusú objektum lesz, melynek adattagjai dinamikusan jönnek létre (pár sorral lentebb látni fogjuk, hogyan).

Az adatok objektum információhordozó adattagját ugyanolyan néven kell létrehozni, mint a PHP szkriptben megadott, POST metódussal beolvasott változó neve. Esetünkben ez az adat nevű adattag lesz ActionScriptben: ez fog a PHP szkriptben mint $adat megjelenni. Először hozzuk létre az objektumot!

```
public var adatok:URLVariables= new URLVariables();
```

A PHP szkriptjeink helyét két URLRequest objektummal adjuk meg. Mivel ezek nem változnak a program futása során, nyugodtan globális objektumokként is használhatjuk őket, a függvényeken kívül.

```
public var olvas_URL:URLRequest=new URLRequest("http://localhost/olvas.php");

public var kiir_URL:URLRequest=new URLRequest("http://localhost/kiir.php");
```

Ennek megfelelően a szkripteket végrehajtó objektumokból is kettőre lesz szükségünk:

```
public var olvas_Loader:URLLoader = new URLLoader();

public var kiir_Loader:URLLoader = new URLLoader();
```

Ez az alkalmazás is rendelkezni fog egy inicializációs függvénnyel, melyben a szkriptek feldolgozásáért felelős függvényeket rendeljük a megfelelő eseményekhez. Itt adjuk meg explicit módon az adatátadás módját is a PHP szkripteknek (POST típusú feldolgozás).

```
protected function init():void {
olvas_URL.method=URLRequestMethod.POST;

kiir_URL.method=URLRequestMethod.POST;

olvas_Loader.addEventListener(IOErrorEvent.IO_ERROR, IO_hiba);
```

```
kiir_Loader.addEventListener(IOErrorEvent.IO_ERROR, IO_hiba);

olvas_Loader.addEventListener(Event.COMPLETE,adat_olvas_befejezes);

kiir_Loader.addEventListener(Event.COMPLETE,adat_kiir_befejezes);
}
```

Amennyiben a szkriptek nem elérhetőek (pl. nincsen internetkapcsolat), akkor egy IO_ERROR esemény generálódik, amit az IO_hiba függvény fog kezelni. A függvény kódja nagyon egyszerű, egyszerűen kiírunk egy hibaüzenetet a „kimeneti" szövegdobozunkba, akárcsak az előző fejezet példaalkalmazásánál.

```
public function IO_hiba(event:IOErrorEvent):void
{
 kimenet.text="Hálózati kommunikációs hiba.";
}
```

A szkriptek futtatásának végén pedig szintén egy-egy esemény, nevezetesen az Event.COMPLETE fog kiváltódni, ami a szkript feldolgozásának sikeres végét jelenti.

Figyelem: Ez nem feltétlenül jelenti a szkriptben leírt műveletek sikerességét is! Csupán arról van szó, hogy a szkript végrehajtását a szerver befejezte. A szkripten belüli műveletek ellenőrzéséről a szkript által visszaadott információk feldolgozásával gondoskodhatunk.

A PHP szkriptek feldolgozása az URLRequest és URLLoader objektumokkal aszinkron módon kell, hogy történjen. Ezért mindkét szkriptünk futtatásához ismét két-két függvényre lesz szükségünk. Egyre a szkriptfuttatás kezdeményezéséhez és egyre a futtatás eredményének a feldolgozásához.

```
public function adat_olvas_kezdes():void
{
	olvas_Loader.load(olvas_URL);
}

public function adat_olvas_befejezes(e:Event):void
{
	kimenet.text=e.target.data;
}
```

```
 public function adat_kiir_kezdes():void                                    kiir_URL.data=adatok;
{
 if(beviteli_mezo.text == "") return;       adatok.adat=beviteli_mezo.text;
kiir_Loader.load(kiir_URL);
}

public function adat_kiir_befejezes(e:Event):void
{
 kimenet.text=e.target.data;
}
```

Az adat_olvas_kezdes() függvény egyszerűen csak futtatási kérést küld a szerverhez. Mivel az olvas.php szkript nem igényel átadott adatot, ezért elég, ha csak lefuttatjuk a szkriptet a szerveren.

Az adat_olvas_befejezes() függvény, ami a szkript lefutásakor hívódik meg, a kimeneti szövegmezőben megjeleníti a szkript kimenetét. A kimenetet a függvénynek átadott „e" paraméter (eseményobjektum) e.target.data adattagja tartalmazza. Ez alapértelmezés szerint szöveges formátumú adat. Így, amennyiben a szkript futtatása alatt hiba következik be, akkor a hibaüzenet (melyeket a PHP szkriptjeinkben implementáltunk az 5.2-es fejezetben) is ugyanitt fog megjelenni.

Lényegében az adat_kiir_kezdes() és adat_kiir_befejezes() függvénypáros is hasonlóan működik. Az első függvényben ellenőrizzük, hogy üres-e a beviteli mező. Üres szöveget az alkalmazás nem enged elküldeni. Továbbá meg kell határoznunk a szkriptnek átadott adatokat. Ezt az alábbi két utasítás végzi el:

```
adatok.adat=beviteli_mezo.text; kiir_URL.data=adatok;
```

Ekkor jön létre, kvázi dinamikusan az adatok objektum adat nevű adattagja.

Külön hibakezelést nem végeznek a szkriptek kimeneteit feldolgozó függvényeink, példaprogramunkban bőségesen elegendő, ha csak kiírjuk a szkriptek által visszaadott adatokat. Bonyolultabb alkalmazásoknál ennél természetesen kifinomultabb hibakezelésekre is szükség lehet.

Az alkalmazást az alábbi függvénnyel zárjuk be:

```
protected function Kilepes():void
{
olvas_Loader.removeEventListener(Event.COMPLETE,adat_olvas_befejezes);
kiir_Loader.removeEventListener(Event.COMPLETE,adat_kiir_befejezes);
olvas_Loader.removeEventListener(IOErrorEvent.IO_ERROR, IO_hiba);
kiir_Loader.removeEventListener(IOErrorEvent.IO_ERROR, IO_hiba);

NativeApplication.nativeApplication.exit();
}
```

Az alkalmazás teljes forráskódja:

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application
xmlns:fx="http://ns.adobe.com/mxml/2009"

      xmlns:s="library://ns.adobe.com/flex/spark"                    applicationDPI="160"
                  applicationComplete="init()">

<fx:Script> <![CDATA[  import mx.events.FlexEvent;

public var adatok:URLVariables= new URLVariables(); //változók a PHP szkripteknek

public var olvas_URL:URLRequest=new URLRequest("http://localhost/teszt/olvas.php");

public var kiir_URL:URLRequest=new URLRequest("http://localhost/teszt/kiir.php");

public var olvas_Loader:URLLoader = new URLLoader();

public var kiir_Loader:URLLoader = new URLLoader(); protected function init():void
{
olvas_URL.method=URLRequestMethod.POST;

      kiir_URL.method=URLRequestMethod.POST;
olvas_Loader.addEventListener(IOErrorEvent.IO_ERROR, IO_hiba);
kiir_Loader.addEventListener(IOErrorEvent.IO_ERROR, IO_hiba);
olvas_Loader.addEventListener(Event.COMPLETE,adat_olvas_befejezes);
kiir_Loader.addEventListener(Event.COMPLETE,adat_kiir_befejezes);
}

protected function Kilepes():void
{
olvas_Loader.removeEventListener(Event.COMPLETE,adat_olvas_befejezes);
kiir_Loader.removeEventListener(Event.COMPLETE,adat_kiir_befejezes);
olvas_Loader.removeEventListener(IOErrorEvent.IO_ERROR, IO_hiba);
kiir_Loader.removeEventListener(IOErrorEvent.IO_ERROR, IO_hiba);
NativeApplication.nativeApplication.exit();
}

public function IO_hiba(event:IOErrorEvent):void
{
kimenet.text="Hálózati kommunikációs hiba.";
}

public function adat_olvas_kezdes():void
{
olvas_Loader.load(olvas_URL);
```

```
}

public function adat_olvas_befejezes(e:Event):void
{
kimenet.text=e.target.data;
}

public function adat_kiir_kezdes():void
{
if(beviteli_mezo.text == "") return; adatok.adat=beviteli_mezo.text; kiir_URL.data=adatok;
kiir_Loader.load(kiir_URL);
}

public function adat_kiir_befejezes(e:Event):void
{
kimenet.text=e.target.data;
}

]]>
</fx:Script>

<s:TextInput id="beviteli_mezo" top="10" left="10" right="10" horizontalCenter="0" prompt="Írjon
be egy adatot!" text="" textAlign="left"/>

<s:Button id="Gomb_kiiras" top="50" left="10" right="10" label="Kiírás" click="adat_kiir_kezdes()"
horizontalCenter="0"/>

<s:TextArea id="kimenet" left="10" right="10" top="200" bottom="10" horizontalCenter="0"/>

<s:Button id="Gomb_olvasas" top="100" left="10" right="10" label="Olvasás" horizontalCenter="0"
click="adat_olvas_kezdes()"/>

<s:Button id="Gomb_Kilep" top="150" left="10" right="10" label="Kilépés" horizontalCenter="0"
click="Kilepes()"/>

</s:Application>
```

Teszteljük alkalmazásunkat!
Figyeljünk a következőkre:

- A három alapfunkció működik-e:
  - o kiírás o
  - olvasás o
  - kilépés.
- Az adatbeírásokat lekérdezésekkel ellenőrizzük az adatbázisban is!
- A szkriptek összes lehetséges kimenete megjelenik-e a szövegdobozban?
- Állítsunk elő mesterségesen hibahelyzeteket! Állítsuk le az Apache szervert, vagy a

MySQL szolgáltatást és nézzünk meg, hogy a megfelelő hibaüzenetek megjelennek-e olvasáskor, és/vagy kiíráskor!

5.4. Az alkalmazás éles üzembe helyezése

Ha minden megfelelően működik, az elkészült alkalmazást publikálhatjuk és az elkészült .APK kiterjesztésű telepítőcsomagot akár el is küldhetjük barátainknak, ismerőseinknek, illetve közzétehetjük a megfelelő webáruházban is.

Éles, online környezet használatakor ne felejtsük el az online szerveren is létrehozni a megfelelő adatbázist és adattáblát, valamint a PHP szkripteket elhelyezni (tippekért ld. 1. fejezetet).

6. Receptek haladóknak

Akik már elkészítették első alkalmazásaikat, azok számára ebben a fejezetben néhány ”instant” kódreceptet tartogatunk.

Ezek még színesebbé (szó szerint is) tehetik alkalmazásainkat és az AIR további önálló felfedezésére ösztönöznek.

6.1. Háttérszín megváltoztatása

Az alkalmazás háttérszíne alapesetben fekete, de ezt megváltoztathatjuk a Flex kód kezdő <s:Apllication> tagjének kiegészítésével.

A backgroundColor tulajdonság egy hexadecimális színértéket vehet fel (sorrendben a vörös, zöld és kék színértékeket megadva).

Az alábbi példában háttérszínnek a 00AA00 értéket adjuk meg, ami egy élénkzöld szín kódja.

```
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"

xmlns:s="library://ns.adobe.com/flex/spark" applicationDPI="160"

backgroundColor="#00AA00">
```

6.2. Betöltőkép megjelenítése

Az ún. 'Splash Screen' az alkalmazás betöltődésekor látható, általában egy logót, az alkalmazás nevét stb. szokták így megjeleníteni.

Splash képet a Flex kód kezdő <s:Apllication> kódblokkjában adhatunk meg. Háromféle tulajdonság tartozik a Splash képernyőkhöz.

SplashScreenImage: A kép erőforrásfájl neve. Egy @Embed('*képfájl_neve*') kifejezéssel hozzá kell csatolnunk az alkalmazásunkhoz.

SplashScreenScaleMode: a képarányok megjelenítésének módja. Négyféle értéke lehet: none, letterbox, stretch, zoom. A két leggyakoribb talán a letterbox és a stretch. Előbbi esetén az eredeti képarányok mindenképpen megmaradnak, viszont lehet, hogy a kép nem tölti ki teljesen a képernyőt. Utóbbi esetében a képernyő kitöltése garantált, viszont a képarányok torzulhatnak. Ha a kép kisebb, mint a képernyő mérete, a zoom értékkel annyira felnagyítódik a kép, hogy kitöltse a teljes képernyőt.

SplashScreenMinimumDisplayTime:
millimásodpercben megadhatjuk, hogy mennyi ideig legyen látható a betöltőkép.

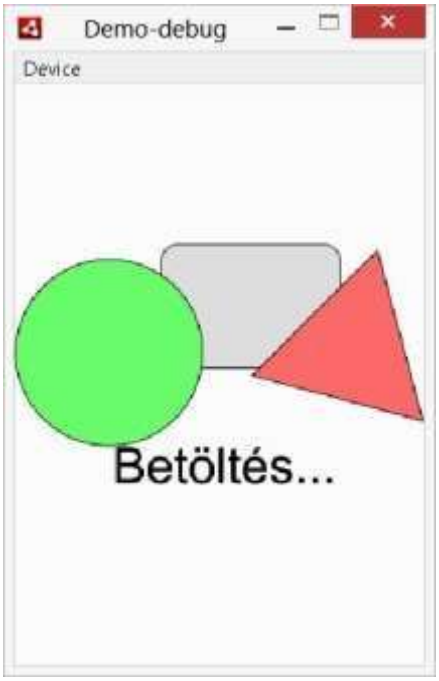
Az alábbi kóddal a splash_screen.png képfájl jelenik meg az alkalmazás betöltésekor, 1 másodpercen keresztül.

```
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
xmlns:s="library://ns.adobe.com/flex/spark" applicationDPI="160"
splashScreenImage="@Embed('src/splash_screen.png')" splashScreenScaleMode="letterbox"
splashScreenMinimumDisplayTime="1000">
```

6.3. Statikus képek megjelenítése

Az Image komponenssel képeket jeleníthetünk meg alkalmazásainkban.

Választhatunk, hogy a képfájl külön fájlként töltődjön be, vagy az alkalmazásba legyen beágyazva. Utóbbi esetben az Embed kulcsszót kell használnunk.



Amennyiben a width és/vagy height tulajdonságokat is megadjuk, a komponens scaleMode tulajdonságával megszabhatjuk a kép megjelenését. Háromféle értéket vehet fel: letterbox, stretch, zoom, melyek a képnek a számára megadott területen történő megjelenését befolyásolják.

Példák:

```
<s:Image x="40" y="40" source="duck.png"/>

<s:Image x="40" y="40"  width="237" scaleMode="letterbox"  source="@Embed('duck.png')"/>
```

6.4. Képernyő méreteinek lekérdezése

Szükségünk lehet a képernyő méreteinek lekérdezésére is.

A használt eszköz képernyőfelbontásának lekérdezése nagyon egyszerű. Az alkalmazás felületének (az AIR fogalomrendszerében ezt stage-nek hívják) két tulajdonságát kell csupán lekérdeznünk, melyek „automatikusan" tartalmazzák az értékeket.

A vízszintes és a függőleges képernyőfelbontást a stage.width és a stage.height tartalmazza.

Ezek decimális egész értékek (int).

Ezeket a tulajdonságokat az ActionScript kód bármely részén lekérdezhetjük.

6.5. Képernyőtájolás kezelése

A képernyőforgatás kezelésére kétféle módszer is rendelkezésre áll.

Az első, egyszerűbb megoldással az alkalmazásunk tájolását előre megszabhatjuk. Ehhez elég az -app.xml alkalmazásleíró fájlt módosítanunk. Két bejegyzés fontos számunkra:

```
<autoOrients></autoOrients>
```

True, vagy false értéket megadva befolyásolhatjuk, hogy a készülék elforgatásakor kérjüke, hogy a képernyőtartalom is átrendeződjön, vagy sem.

```
<aspectRatio></aspectRatio>
```

Portrait, vagy Landscape értéket megadva az alkalmazásunk alapértelmezett képernyőelforgatását adhatjuk meg.

Természetesen mindkét érték beállításának csak az alkalmazásunkra lesz hatása.

A második módszerrel ActionScript kódból kezeljük a képernyőtájolást.

Ehhez szintén a Stage osztályt használjuk, mégpedig a setAspectRatio metódusát.

A két lehetséges képernyőtájolás beállítása a következőképpen történhet:

```
stage.setAspectRatio(StageAspectRatio.PORTRAIT);

stage.setAspectRatio(StageAspectRatio.LANDSCAPE);
```

6.6. Teljesképernyős üzemmód

Alkalmazásunk megjelenítését szükség esetén teljesképernyős üzemmódba is állíthatjuk. Teljesképernyős üzemmód alatt azt kell érteni, hogy alkalmazásunk felülete kitakarja a mobileszköz kijelzőjének alsó (vagy felső) részén található státuszsort is (ebben a sorban jelenik meg pl. az akkumulátor töltöttségi szintje, az idő stb.).

Az üzemmódváltáshoz az alapértelmezetten rendelkezésre álló stage osztály displayState tulajdonságát kell módosítanunk.

Az alábbi utasítással teljesképernyős módra váltunk:

```
stage.displayState=StageDisplayState.FULL_SCREEN;
```

Ebben az esetben a billentyűzet interaktivitás kikapcsolt állapotba kerül. Ha az Android virtuális billentyűzetét teljesképernyős módban is elérhetővé akarjuk tenni, a következő módon váltsunk teljes képernyőre:

```
stage.displayState=StageDisplayState.FULL_SCREEN_ INTERACTIVE;
```

A teljesképernyős megjelenítésről a következőképpen válthatunk vissza normál képernyőmódba:

```
stage.displayState=StageDisplayState.NORMAL;
```

6.7. SMS küldés kezdeményezése

Amennyiben mobileszközünk SMS küldését is támogatja, egyszerűen felhívhatjuk az alapértelmezett SMS-küldő szoftvert, sőt, még a címzett telefonszámot is megadhatjuk.

Ehhez egy URLRequest objektumot kell készíteni és paraméterként átadni a navigateToURL beépített ActionScript függvénynek.

Az URL lényegében egy ”sms:” kezdetű sztring, melyben a kettőspont után megadhatunk egy telefonszámot.

Az alábbi példa a 061123456 telefonszámra küldendő új üzenet megszerkesztéséhez indítja el a telefon SMS küldő szoftverét:

```
public var s:String;

s+="sms:061123456"; navigateToURL(new URLRequest(s));
```

Megjegyzendő, hogy az utasítással az SMS-t nem küldjük el, csupán a telefon alapértelmezett SMS-küldő alkalmazását „paraméterezzük fel” és indítjuk el alkalmazásunkból. Amennyiben

több szövegküldő alkalmazás is telepítve van a mobileszközre, kiválaszthatjuk, melyiket kívánjuk használni.

Miután megírtuk az SMS-t, visszajutunk alkalmazásunkba. Amennyiben megszakítjuk a műveletet, akkor is az alkalmazásunkhoz kerül vissza a vezérlés.

6.8. Telefonhívás kezdeményezése

Az előző alfejezetben leírthoz hasonlóan telefonhívást is kezdeményezhetünk.

Ekkor az URL-ben a ”tel:” szót kell használnunk. A kettőspont után itt is megadhatunk egy telefonszámot.

Az alábbi példa a 061123456 telefonszámot előre megadva hívja fel a telefon alapértelmezett tárcsázószoftverét:

```
public var s:String;

s+="tel:061123456"; navigateToURL(new URLRequest(s));
```

A módszerrel a telefonszám tárcsázása nem történik meg, csupán a telefon alapértelmezett tárcsázó alkalmazását „paraméterezzük fel” és indítjuk el alkalmazásunkból.

6.9. Email küldés kezdeményezése

A 6.7 és 6.8 alfejezetekben ismertetett módon email írását is kezdeményezhetjük alkalmazásunkból.

Ekkor az URL-ben a ”mailto:” szót kell használnunk. A kettőspont után előre megadhatjuk az email tárgyát és szövegtörzsét is.

Az alábbi példa az xyz@valami.com email címet címzettként megadva, a tárgy mezőt a ”Tárgy” és a levéltörzset az ”Üzenet”szöveggel kitöltve indítja el a telefon alapértelmezett levelezőszoftverét:

```
public var s:String;

s+="mailto:xyz@valami.com?subject=Tárgy&body= Üzenet";  navigateToURL(new URLRequest(s));
```

A módszerrel a levél elküldése nem történik meg, csupán a telefon alapértelmezett levelező alkalmazását „paraméterezzük fel” és indítjuk el alkalmazásunkból. Amennyiben több levélküldő alkalmazás is telepítve van a mobileszközre, kiválaszthatjuk, melyiket kívánjuk használni.

Tapasztalat szerint a módszer a Gmail mobilkliensével működik legteljeskörűbben.

6.10. URL megnyitásának kezdeményezése

A 6.7-6.9 alfejezetekben megismert módszerhez nagyon hasonlóan indíthatjuk el alkalmazásainkból az Android alapértelmezett böngészőjét és kereshetünk fel automatikusan egy webhelyet (ha több böngésző is telepítve van, futás közben kiválaszthatjuk, melyiket akarjuk használni). Itt is a navigateToURL függvényt használjuk.

Például a Google kezdőoldalának az Android külső böngészőprogramjával történő megtekintéséhez az alábbi kódrészletet használhatjuk:

```
public var s:String;

s+="http://www.google.com"; navigateToURL(new URLRequest(s));
```

6.11. Böngésző integrálása mobilalkalmazásba

Felmerülhet az igény, hogy alkalmazásainkon belül, abba integrálva jeleníthessünk meg webes tartalmakat, akár egész oldalakat. Az AIR erre is kínál lehetőséget, mégpedig a StageWebView osztályon keresztül.

A StageWebView lényegében az Android alapértelmezett böngészőmotorját használja az AIRen keresztül.

A webes tartalmak megjelenítése kiváló minőségű, ám az automatikus kicsinyítés/nagyítás csak azoknál a weboldalaknál működik, amelyeket eleve felkészítettek mobileszközön történő megjelenítésre. Ha ez nem így van, akkor sincsen semmi probléma, de például egy 1024x768as felbontásra optimalizált weboldalt egy 480x320-as felbontású mobiltelefon képernyőjén nem fog kiférni, hanem sokat kell görgetnünk a képernyőt.

A StageWebView használata előtt meg kell győződnünk róla, hogy az adott eszköz biztosan támogatja-e a megjelenítést. Ezt a

StageWebView.isSupported tulajdonság ellenőrzésével érhetjük el (true az értéke, ha támogatott az osztály).

Egy böngészőt alkalmazásunk felületén az alábbi utasításokkal hozhatunk létre:

```
public var bongeszo:StageWebView;

bongeszo = new StageWebView(); bongeszo.stage = this.stage;

bongeszo.viewPort = new Rectangle(x,y,width,height);
```

Az objektumpéldányt hozzá kell adnunk a stage-hez, majd a megjelenítéshez használni kívánt területet is meg kell adnunk, hogy lássuk is a webes tartalmakat.

Weblap betöltéséhez a LoadURL metódust használhatjuk. Ügyelnünk kell rá, hogy a címek tartalmazzák a ”http://” előtagot, ellenkező esetben az oldal nem fog betöltődni és hibaüzenetet kapunk.

A StageWebView az IOErrorEvent és az ErrorEvent osztályokban definiált hibaeseményeket is kiválthatja, ezért ezek kezelésére érdemes hibakezelő függvényt írnunk.

Amennyiben bármilyen okból el kell rejtenünk a böngészőt, azt a következő utasítással tehetjük meg:

```
free_bongeszo.viewPort=null;
```

Az újbóli megjelenítéshez ismét definiálnunk kell a megjelenés területét a fentebb ismertetett módon.

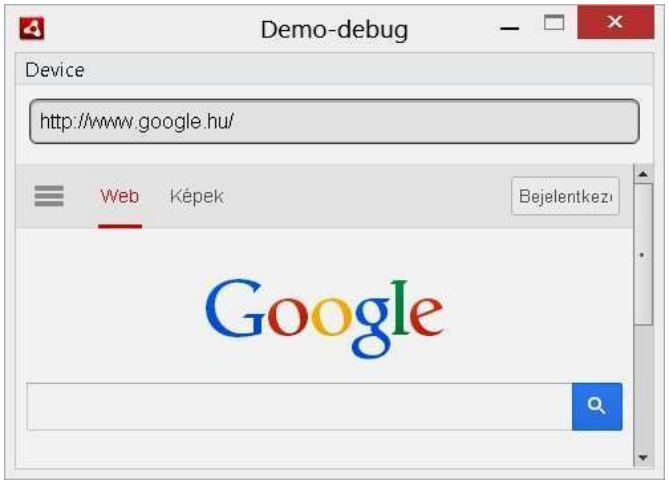
Lehetőség van arra is, hogy a böngészés során az aktuálisan betöltött weboldal linkjét lekérdezzük, például azért, hogy ezt is megjeleníthessük. Ezt egy speciális esemény figyelésével, a LocationChangeEvent.LOCATION_CHANGE esemény kezelésével valósíthatjuk meg.

Az eseménykezelő függvény az event:LocationChangeEvent függvényargumentumon keresztül, az event.location
tulajdonsággal kaphatja meg az aktuális címet, amikor az megváltozik.

A fentiek illusztrálására lássuk egy teljes kisalkalmazás kódját, mely induláskor a Google.hu weboldalát tölti be. Az alkalmazás felső részén egy szövegdobozban mindig megjelenik az aktuálisan böngészett/betöltött oldal. A hibák kezelésétől most eltekintünk.

```
<?xml version=”1.0” encoding=”utf-8”?>
<s:Application
xmlns:fx=http://ns.adobe.com/mxml/2009

xmlns:s=”library://ns.adobe.com/flex/spark” applicationDPI=”160”

backgroundColor=”#FFFFFF”

applicationComplete=”init(event)”>

<fx:Script> <![CDATA[ import mx.events.FlexEvent; public var bongeszo:StageWebView;

protected function init(event:FlexEvent):void
{
if(StageWebView.isSupported== true)
{
bongeszo = new StageWebView(); bongeszo.stage = this.stage;

bongeszo.addEventListener(LocationChangeEvent.LOCATION_CHANGE,frissit);
```

```
}

webhely.text=”google.hu”;

bongeszo.viewPort = new
Rectangle(0,60,stage.width,stage.height-60);  bongeszo.loadURL(„http://google.hu”);
}

public function frissit(event:LocationChangeEvent): void
{
webhely.text=event.location;
}

]]> </fx:Script>

<s:TextInput id=”webhely” left=”10” right=”10” top=”10”/>

</s:Application>
```



6.12. Busy indicator használata

A felhasználók felé kényelmes módon jelezhetjük azt, hogy alkalmazásunk műveletet végez. Erre a célra használhatjuk a BusyIndicator komponenst, ami lényegében a Windows homokórájának androidos megfelelője. Androidon nem homokórát, hanem egy körbe-körbe forgó pálcikát jeleníthetünk meg.

Pusztán a felhasználó felé történő visszajelzést szolgálja, kihatása az alkalmazások működésére nincsen.

Két tulajdonságát emeljük ki.

RotationInterval: a forgás sebességét változtathatjuk meg az alapértékről. Milliszekundum értéket kell megadni.

SymbolColor: A szimbólum színét változtathatjuk meg. Hexadecimális formában kell megadnunk, pl.: #00FF00.

A BusyIndicator-t nem lehet a szó valódi értelmében elindítani, vagy leállítani. A használata úgy történik, hogy a visible tulajdonságának true, vagy false értékre állításával megjelenítjük, vagy eltávolítjuk a képernyőről. Az animáció automatikus.

6.13. Hangfájlok lejátszása

Hangfájlok egyszerű lejátszására az URLRequest és a Sound osztályokat használhatjuk.

Az alábbi kódrészlet a 'zene.mp3' zenefájlt lejátszását mutatja be.

```
public var zenefajl:URLRequest = new URLRequest("zene.mp3");

public var lejatszo:Sound = new Sound(zenefajl);

s.play();
```

7. Hogyan tovább?

Most, hogy a Kedves Olvasó már rendelkezik a szükséges alapvető ismeretekkel az ActionScript- és Flex-alapú mobilfejlesztéshez, felmerülhet a kérdés: „Hogyan tovább?”

Tisztázni kell magunkban, hogy milyen jellegű alkalmazásokat szeretnénk készíteni.

Ezek lehetnek üzleti alkalmazások, segédprogramok, játékprogramok stb. Az Adobe AIR segítségével mindent elérhetünk, csak kitartásra van szükségünk. A könyv irodalomjegyzéke további segítséget kíván nyújtani a továbblépéshez. Rengeteg weboldal és még több szakkönyv elérhetőségét ill. pontos megnevezését találhatjuk a könyv végén. Ezek közül sok forrást a szerző is rendszeresen használ napi szinten.

Függelék

Egy alkalmazásleíró XML fájl teljes tartalma:

```
<?xml version="1.0" encoding="utf-8" standalone="no"?> <application
xmlns="http://ns.adobe.com/air/application/3.1">

<!-- Adobe AIR Application Descriptor File Template.

Specifies parameters for identifying, installing, and launching AIR applications.
  xmlns - The Adobe AIR
namespace:
http://ns.adobe.com/air/application/3.1
The last segment of the namespace specifies the version of the AIR runtime required
for this application to run.

minimumPatchLevel - The minimum patch level of the
AIR runtime required to run the application. Optional.
-->
   A universally unique application identifier. Must be unique across all AIR
applications. Using a reverse DNS-style name as the id is  recommended. (Eg.
com.example.ExampleApplication.) Required. -->

<id>FeherKrisztian.Pelda1</id>

<!-- Used as the filename for the application. Required. -->
<filename>Pelda1</filename>

<!-- The name that is displayed in the AIR application installer.   May
have multiple values for each language. See
samples or xsd schema file. Optional. --> <name>Pelda1</name>

<!-- A string value of the format <0-999>.<0-999>.<0999> that represents
application version which can be used to check for application upgrade.
Values can also be 1-part or 2-part. It is not necessary to have a 3-part value.
An updated version of application must have a versionNumber value higher than the
previous version.
Required for namespace >= 2.5 . -->
<versionNumber>0.0.0</versionNumber>

<!-- A string value (such as "v1", "2.5", or "Alpha 1") that represents the version
of the application, as it should be shown to users. Optional. --> <!--
<versionLabel></versionLabel> -->

   Description, displayed in the AIR application installer.
May have multiple values for each language. See samples or xsd schema file.
Optional. --> <!-- <description></description> -->

<!-- Copyright information. Optional --> <!-- <copyright></copyright> -->

<!-- Publisher ID. Used if you're updating an application created prior to 1.5.3 ->
<!-- <publisherID></publisherID> -->
```

```
<!-- Settings for the application's initial window.  Required.
-->
<initialWindow>
<!-- The main SWF or HTML file of the application.
Required. -->
<!-- Note: In Flash Builder, the SWF reference is set automatically. -->
<content>[This value will be overwritten by Flash Builder in the output
app.xml]</content>


<!-- The title of the main window. Optional. --> <!-- <title></title> -->


<!-- The type of system chrome to use (either "standard" or "none"). Optional.
Default standard. -->
<!-- <systemChrome></systemChrome> -->


<!-- Whether the window is transparent. Only applicable when systemChrome is none.
Optional.
Default false. -->
   <transparent></transparent> -->


<!-- Whether the window is initially visible. Optional.
Default false. -->
<!-- <visible></visible> -->


<!-- Whether the user can minimize the window.
Optional. Default true. -->
<!-- <minimizable></minimizable> -->


<!-- Whether the user can maximize the window.
Optional. Default true. -->
<!-- <maximizable></maximizable> -->


<!-- Whether the user can resize the window. Optional.
Default true. -->
<!-- <resizable></resizable> -->


<!-- The window's initial width in pixels. Optional. --> <!-- <width></width> -->


<!-- The window's initial height in pixels. Optional. --> <!-- <height></height> -
->


<!-- The window's initial x position. Optional. --> <!-- <x></x> -->


<!-- The window's initial y position. Optional. --> <!-- <y></y> -->


<!-- The window's minimum size, specified as a width/ height pair in pixels, such
as "400 200". Optional. --> <!-- <minSize></minSize> -->


   The window's initial maximum size, specified as a width/height pair in pixels,
such as "1600 1200".
Optional. -->
<!-- <maxSize></maxSize> -->
```

```
<!-- The initial aspect ratio of the app when launched (either "portrait" or
"landscape"). Optional. Mobile only. Default is the natural orientation of the
device -->

<!-- <aspectRatio></aspectRatio> -->

<!-- Whether the app will begin auto-orienting on launch. Optional. Mobile only.
Default false -->

<!-- <autoOrients></autoOrients> -->

<!-- Whether the app launches in full screen. Optional. Mobile only. Default false
-->

<!-- <fullScreen></fullScreen> -->

<!-- The render mode for the app (either auto, cpu, gpu, or direct). Optional.
Default auto -->

<!-- <renderMode></renderMode> -->

<!-- Whether or not to pan when a soft keyboard is raised or lowered (either "pan"
or "none").  Optional.
Defaults "pan." -->
<!-- <softKeyboardBehavior></softKeyboardBehavior> -->
<autoOrients>true</autoOrients>
<fullScreen>false</fullScreen>
<visible>true</visible>

<softKeyboardBehavior>none</softKeyboardBehavior> </initialWindow>

<!-- We recommend omitting the supportedProfiles element, -->
<!-- which in turn permits your application to be deployed to all -->
<!-- devices supported by AIR. If you wish to restrict deployment -->
<!-- (i.e., to only mobile devices) then add this element and list -->
<!-- only the profiles which your application does support. -->  <!--
<supportedProfiles>desktop extendedDesktop mobileDevice
extendedMobileDevice</supportedProfiles> -->

<!-- The subpath of the standard default installation location to use. Optional.
--> <!-- <installFolder></installFolder> -->

<!-- The subpath of the Programs menu to use. (Ignored on operating systems without
a Programs menu.)
Optional. -->
<!-- <programMenuFolder></programMenuFolder> -->

<!-- The icon the system uses for the application. For at least one resolution,
specify the path to a PNG file included in the AIR package. Optional. -->  <!--
<icon>
 <image16x16></image16x16>
 <image32x32></image32x32>
 <image36x36></image36x36>
 <image48x48></image48x48>
 <image57x57></image57x57>
```

```
 <image72x72></image72x72>
 <image114x114></image114x114>
 <image128x128></image128x128>
</icon> -->

<!-- Whether the application handles the update when a user double-clicks an update
version of the AIR file (true), or the default AIR application installer handles
the update (false).
Optional. Default false. -->
<!-- <customUpdateUI></customUpdateUI> -->

<!-- Whether the application can be launched when the user clicks a link in a web
browser.
Optional. Default false. -->
<!--
<allowBrowserInvocation></allowBrowserInvocation> - ->

<!-- Listing of file types for which the application can register. Optional. -->
<!-- <fileTypes> -->

<!-- Defines one file type. Optional. --> <!-- <fileType> -->

<!-- The name that the system displays for the
registered file type. Required. --> <!-- <name></name> -->
<!-- The extension to register. Required. --> <!-- <extension></extension> --
>

<!-- The description of the file type. Optional. --> <!--
<description></description> -->

<!-- The MIME content type. -->
<!-- <contentType></contentType> -->

<!-- The icon to display for the file type. Optional. -->  <!--
<icon>
 <image16x16></image16x16>
 <image32x32></image32x32>
 <image48x48></image48x48>
 <image128x128></image128x128>
</icon> -->

<!-- </fileType> -->
<!-- </fileTypes> -->

<!-- iOS specific capabilities -->
<!-- <iPhone> -->
<!-- A list of plist key/value pairs to be added to the application Info.plist -->
<!-- <InfoAdditions>
<![CDATA[
<key>UIDeviceFamily</key>
<array>
<string>1</string>
<string>2</string>
</array>
<key>UIStatusBarStyle</key>
<string>UIStatusBarStyleBlackOpaque</string>
```

```
<key>UIRequiresPersistentWiFi</key>
<string>YES</string>  ]]>
</InfoAdditions> -->
<!-- A list of plist key/value pairs to be added to the application
Entitlements.plist -->
<!-- <Entitlements>
<![CDATA[
    <key>keychain-access-groups</key>
       <array>
             <string></string>
             <string></string>
       </array>
    ]]>
</Entitlements> -->
<!-- Display Resolution for the app (either "standard" or
"high"). Optional. Default "standard" -->
<!--
<requestedDisplayResolution></requestedDisplayResol ution> --> <!-- </iPhone> -->


<!-- Specify Android specific tags that get passed to
AndroidManifest.xml file. -->
<!--<android> -->
<!-- <manifestAdditions>
<![CDATA[
<manifest android:installLocation="auto">
<uses-permission
android:name="android.permission.INTERNET"/>
<uses-permission
android:name="android.permission.WRITE_EXTERNAL_ STORAGE"/>
<uses-permission android:name="android. permission.ACCESS_FINE_LOCATION"/>
<usesfeature android:required="true"
android:name="android.hardware.touchscreen.multitouch"/>
<application android:enabled="true">
<activity android:excludeFromRecents="false">
<intent-filter>
<action android:name="android.intent.action.MAIN"/>
<category
android:name="android.intent.category.LAUNCHER"/>
</intent-filter>
</activity>
</application>
</manifest>
]]>
</manifestAdditions> -->
<!-- Color depth for the app (either "32bit" or "16bit"). Optional. Default 16bit
before namespace 3.0, 32bit after -->
<!-- <colorDepth></colorDepth> -->
<!-- </android> -->
<!-- End of the schema for adding the android specific tags in AndroidManifest.xml
file -->
<android>
<colorDepth>16bit</colorDepth>
<manifestAdditions><![CDATA[
<manifest android:installLocation="auto"> <!--See the Adobe AIR documentation for
more information about setting Google Android permissions-->
<!--Removing the permission android.permission.INTERNET will have the side effect
of preventing you from debugging your application on your device--> <usespermission
android:name="android.permission.INTERNET"/>
<!--<uses-permission
android:name="android.permission.WRITE_EXTERNAL_
STORAGE"/>-->  <!--<uses-permission
```

```
android:name="android.permission.READ_PHONE_STA
TE"/>-->
<!--<uses-permission
android:name="android.permission.ACCESS_FINE_LOC
ATION"/>-->
<!--The DISABLE_KEYGUARD and WAKE_LOCK permissions should be toggled together in
order to access AIR's SystemIdleMode APIs-->
<!--<uses-permission
android:name="android.permission.DISABLE_KEYGUAR
D"/>-->
<!--<uses-permission
android:name="android.permission.WAKE_LOCK"/>-->
<!--<uses-permission
android:name="android.permission.CAMERA"/>--> <!--<uses-permission
android:name="android.permission.RECORD_AUDIO"/> -->
<!--The ACCESS_NETWORK_STATE and
ACCESS_WIFI_STATE permissions should be toggled together in order to use AIR's
NetworkInfo APIs-->
<!--<uses-permission
android:name="android.permission.ACCESS_NETWORK
_STATE"/>-->  <!--<uses-permission
android:name="android.permission.ACCESS_WIFI_STAT E"/>-->
</manifest>


]]></manifestAdditions>
</android>
<iPhone>
        <InfoAdditions><![CDATA[
     <key>UIDeviceFamily</key>
      <array>
       <string>1</string>
       <string>2</string>


      </array>
   ]]></InfoAdditions>

<requestedDisplayResolution>high</requestedDisplay
Resolution>
    </iPhone>
</application>
```

Ajánlott irodalom

Flex és Actionscript

Adobe Creative Team: Adobe Flash CS4 Professional Tanfolyam a könyvben. (Adobe Press), Perfact-Pro Kft., 2009.
Adobe Creative Team: ActionScript 3.0 Adobe
Flash Professional alkalmazáshoz. (Adobe Press),
Perfact-Pro Kft., 2009.
Colin Moock: ActionScript 3.0 a gyakorlatban. Kiskapu kft., 2008.
Michael Labriola, Jeff Tapper, Matthew Boles: Adobe Flex 4.5 Fundamentals. Adobe Press, 2011.
Rich Tretola: Developing Applications with Flex 4.5. O'Reilly Media, 2011.
Joseph Labrecque: What's new in Adobe AIR 3. O'Reilly Media, 2012.
Thibault Imbert: Introducing Starling. O'Reilly Media, 2012.
Vérinique Brossier: Developing Android Applications with adobe AIR. O'Reilly Media, 2011.
Scott Citron, Michael Murphy: Adobe Creative
Suite 5 Design Premium HOW-TOs. Adobe Press,
2010.

Androidos fejlesztés és dizájn

Ekler Péter, Fehér Marcell, Forstner Bertalan, Kelényi Imre: Android-alapú szoftverfejlesztés. SZAK Kiadó, 2012.
Uwe Post: Android-Apps entwickeln. Galileo Press, 2013.
Wei-Meng Lee: Android-Bausteine. WILEYVCH Verlag GmbH & Co. KgaA, 2013. Theresa
Neil: Mobile Design Pattern Gallery. O'Reilly Media, 2012.

Apache, MySQL és PHP

Ken Coar, Rich Bowen: Apache receptek. Kiskapu Kft, 2008.
Tricia Ballad & William Ballad: Biztonságos webalkalmazások PHP nyelven. Kiskapu Kft, 2010.
Zak Greant, Chris Newman: MySQL zsebkönyv. Kiskapu Kft, 2007.
Kozmajer Viktor: PHP és MySQL az alapoktól. BBS-INFO Kiadó, 2011.
Sági Gábor: Webes adatbázis-kezelés MySQL és PHP használatával. BBS-INFO Kiadó, 2005.
Luke Welling, Laure Thomson: PHP és MySQL webfejlesztőknek. Perfact-Pro Kft., 2012. Mike
Andrews, James Whittaker: Hogyan törjünk fel webhelyeket? Kiskapu Kft, 2007.

Webhelyek

Google Play Áruház https://play.google.com/store
Amazon Appstore for Android http://www.amazon.com/mobileapps/b?node=2350149011
Az Android-ról
http://developer.android.com/about/index.html
Open Handset Alliance http://www.openhandsetalliance.com Adobe
AIR SDK
http://www.adobe.com/devnet/air/air-sdkdownload.html
Adobe Gaming http://gaming.adobe.com/
Adobe AIR fejlesztői portál http://www.adobe.com/devnet/air.html Adobe Developer Connection
http://www.adobe.com/devnet.html
Adobe AIR runtime (PC/MAC) http://get.adobe.com/air/
Apache Flex http://flex.apache.org
„Flex in a week" ingyenes videotréningsorozat
http://www.adobe.com/devnet/flex/videotraining.html További
Adobe tréningek http://tv.adobe.com

Játékfejlesztéshez

Starling 2D keretrendszer Flash-hez http://gamua.com/starling/

Away3D keretrendszer Flash-hez http://away3d.com/

MySQL http://www.mysql.com

PHP http://php.net
Webhelyek
Apache http://www.apache.org

Stackoverflow (fejlesztői fórum és tudástár) http://stackoverflow.com

Teszteszközök Adobe AIR fejlesztésekhez

https://github.com/bigfish
http://gskinner.com/blog/archives/2010/02/performancetest.html
http://jpauclair.net/2010/12/23/complete-flashprofiler-its-getting-serious/

Angol nyelvű szakirodalom

Adobe Press http://www.adobepress.com

O'Reilly Media http://oreilly.com