**Aantekeningen bij iOS SDK Training**

**Hoofdstuk 0 Introductie**

Deze cursus behandelt de 20% belangrijkste zaken die nodig zijn om iOS applicaties te ontwikkelen, die in vrijwel alle iOS applicaties terugkomen. Verder wordt de eenvoudigste manier om doelen te bereiken toegepast, vrijwel altijd zijn er echter meerdere manieren om een doel te bereiken.

Benodigde voorkennis voor deze training bestaat uit algemene (object oriented) programmeerervaring of – nog beter – ervaring met Objective C. Zie de aparte training daarvoor. In deze cursus wordt kort ingegaan op Objective C.

**Hoofdstuk 1 Opstarten**

De training gaat uit van OS X Lion (of hoger) en XCode 4.2 (of hoger). Let op dat XCode vanaf versie 4.3 vanuit de AppStore gedownload kan worden en zich als een standaard applicatie installeert in de Applicaties folder. Tot en met versie 4.2 werd XCode in een aparte Developer folder in de root geinstalleerd.

Op de website van Apple is up-to-date informatie beschikbaar voor XCode ontwikkelaars. Om hierbij te kunnen is het noodzakelijk je als Apple Developer aan te melden. Apple biedt diverse ontwikkelaar programma’s aan. De gratis variant geeft je toegang tot alle aanwezige documentatie, developer tools en fora. Maar daarmee kun je je zelf ontwikkelde iOS programma alleen met de simulator testen die bij XCode wordt bijgeleverd. Wil je eigen code deployen op een iPhone of iPad dan moet dat via het betaalde Apple Developer programma, dit kost 99$ per jaar. Daarnaast kun je je programma’s ook in de Apple AppStore zetten. Let op dat het een paar dagen duurt na aanmelden voordat je daadwerkelijk kunt deployen.

**1.4 Een eenvoudige iOS applicatie creeren**

We beginnen met het creeren van een ‘Hello world’ programma in iOS. Start XCode en selecteer ‘Create a new XCode project’. Selecteer in het volgende window ‘Application’ onder ‘iOS’ (links in het window) en ‘Single View Application’ (rechts in het window).

Er verschijnt nu een volgend window waarin gegevens over het project ingevoerd moeten worden. Geef als product name ‘HelloWorld’ in (geen spaties!). Als company identifier wordt gewoonlijk de laatste 2 woorden van de eigen URL in omgekeerde volgorde ingegeven (bijv. com.lynda in deze cursus). Deze komt voor de product name (HelloWorld) en vormt zo tesamen de applicatie naam zoals deze in de AppStore zichtbaar zou zijn, in dit geval dus ‘com.lynda.HelloWorld’, dit wordt in XCode aangeduid als Bundle Identifier.

De Class Prefix staat default op XYZ en wordt door XCode aan de naam van eigen gedefinieerde klassen toegevoegd, in dit stadium doen we er niets mee, dus laten we het staan.

Bij Device Family kiezen we nu voor de eenvoudigste keuze ‘iPhone’. De andere keuzes iPad en Universal maken het project wat complexer en dat bewaren we voor later.

Van de drie keuze hieronder alleen ‘Use Automatic Reference Counting’ aanvinken, de andere twee opties voegen extra complexiteit toe die we nu niet nodig hebben. Automatic Reference Counting (ARC) is nieuw in iOS 5.0 en maakt memory management eenvoudiger, er hoeft minder code voor geschreven te worden. In deze cursus gebruiken we het altijd.

Klik nu op Next. Er verschijnt een volgend window waarin moet worden aangegeven waar het project moet worden opgeslagen. Voer dit in. Laat de optie ‘Create local git repository for this project’ onaangevinkt, voor dit project gebruiken we geen source control. XCode kan overigens zowel met Subversion als git omgaan.

Klik op Next en het project wordt gecreeerd. In de Project Navigator zijn de files te zien die XCode voor het project heeft gecreeerd. Klik nu op de ViewController.xib file (iPhone versie, in mijn project is er ook een iPad variant gecreeerd, maar die gebruiken we nu niet). De .xib file is een user interface file. Selecteer je deze file dan wordt een user interface editor geopend met een blanco canvas. Hierop kunnen user interface elementen geplaatst worden vanuit de Object library. Deze is rechts onderin het XCode window zichtbaar en kan worden opgeroepen vanuit het XCode hoofdmenu via View-Utilities-Show Object Library. Selecteer een Label uit de Object Library en sleep dit naar het iPhone canvas. Klik erop en vul de tekst ‘Hello World’ in.

Klik nu op de Run button. Het project wordt gebuild en vervolgens gedraaid. De iPhone simulator verschijnt nu met ons fantastische programma dat de tekst ‘Hello World’ op het schermpje laat zien. De iPhone simulator werkt verder als een iPhone. In een woord fantastisch! Stop de iOS simulator vanuit het OSX menu (Stop iOS simulator).

**1.5 De vier pilaren van software ontwikkeling**

Het ontwikkelen van een iOS applicatie vraagt kennis op een groot aantal gebieden: Objective C, XCode, memory management, MVC, etc etc. Om een goed begin te maken onderscheid Apple 4 pilaren waarop de iOS software ontwikkeling gebouwd is:

* Tools: XCode, iOS simulator en Instruments
* Taal: Objective C, inclusief de ontwerp patronen zoals MVC en delegation, en de ondersteunende frameworks van Apple zoals Cocoa Touch
* Ontwerp: Dit betreft vnl de user interface waarvoor Apple de zg. Human Interface Guidelines heeft opgesteld. Om in de AppStore te worden opgenomen moet de applicatie daar aan voldoen
* Proces: het proces van software ontwikkeling mbt testen, provisioning, signing en submitting (aan de AppStore)

**Hoofdstuk 2 De tools**

**2.1 XCode gebruiken**

De video geeft een overzicht van de mogelijkheden van XCode, de user interface, instellingen, etc. Dit hebben we in de Objective C cursus allemaal gezien en kan in de aantekeningen van de cursus opgezocht worden.

**2.2 De iOS simulator**

Met de iOS simulator kunnen iPhone en iPad applicaties getest worden. De simulator wordt automatisch door XCode opgestart na een succesvolle build. Er zijn echter een aantal beperkingen.

Zo heeft de simulator geen camera (de webcam van de pc wordt er niet voor gebruikt). Er is ook geen accelerometer. De weergegeven iPhone/iPad kan wel gedraaid worden vanuit het simulator Hardware menu. Een schud simulatie kan ook vanuit dit menu.

Na een tijdje programmeren zal de simulator vol zitten met testapplicaties. Deze kunnen verwijderd worden zoals op de iPhone zelf door op een applicatie te klikken en de vast te houden totdat het kruisje X verschijnt. Het is ook mogelijk om de simulator vanuit het iOS simulator menu in een keer naar zijn begintoestand terug te brengen, mbv de Reset optie.

**Hoofdstuk 3 Objective C opfrisser**

Zie hiervoor de aantekeningen van de Objective C cursus. Alleen aanvullingen en wijzigingen worden hier behandeld.

Automatic Reference Counting (ARC)

Sinds de komst van iOS 5.0 en XCode 4.2 is er een belangrijke wijziging:

ARC = Automatic Reference Counting

Wanneer een project wordt gecreeerd is dit een van de project parameters die in XCode kan worden ingesteld. Standaard staat deze aangevinkt.

Met Automatic Reference Counting hoeft geen rekening meer te worden gehouden met memory management. Dit neemt de LLVM compiler voor zijn rekening. Was het voorheen nog nodig om elk object dat niet meer werd gebruikt te releasen, met ARC aangezet neemt de compiler dit voor zijn rekening. Staat ARC aan dan is het zelfs niet meer mogelijk om zelf objecten vrij te geven dmv. een release message, de compiler geeft in dat geval een foutmelding. Hiermee is een belangrijk bezwaar tegen het gebruik van Objective C weggenomen.

@property attributen

Bij de definitie van een property – bedoeld om toegang tot instance variabelen te krijgen – kunnen attributen meegegeven worden. Voor iOS programma’s zijn dat meestal de volgende:

* Nonatomic (vs atomic), houdt geen rekening mee dat meerdere threads toegang tot de instance variabele verkrijgen,
* Strong (vs weak), dit is gerelateerd aan ARC en nieuw in iOS. Het houdt in dat de klasse een sterke relatie heeft met de instance variabele waarvoor de property geldt, wat weer betekent dat zolang een object van de klasse bestaat, de instance variabele ook bestaat, maw dat het geheugen ervoor gereserveerd blijft. Voorheen werd hier wel het retain attribuut voor gebruikt, maar strong is tegenwoordig gebruikelijker.

Voorbeeld:

@property (nonatomic, strong) NSString \*name;

De NSString variabele ‘name’ heeft gegarandeerd een waarde zolang het klasse object bestaat.

**Hoofdstuk 4 IOS project kernvaardigheden**

**4.1 De verschillende iOS project typen gebruiken**

Bij het creeren van een nieuw project biedt XCode verschillende standaard project templates (onder iOS – Application):

* Master Detail application: zoals de Instellingen applicatie, waarbij achter elke optie een nieuwe lijst met opties en/of informatie schuilt. Met de ‘back’ knop boven aan het schermpje navigeer je een niveau terug, etc.;
* OpenGL Game: lijkt op het Game Center. Wordt gebruikt om grafisch zwaardere applicaties te maken mbv de OpenGL bibliotheek. Wordt in deze cursus niet nader toegelicht;
* Page-Based Application: een applicatie waarmee door verschillende pagina’s gebladerd kan worden, mbv de ‘blader’ animatie, alsof je met de hand een pagina omslaat;
* Single View Application: geeft een blanco canvas waarop je zelf de user interface kunt vormgeven;
* Tabbed Application: hiermee kun je een applicatie maken die meerdere pagina’s beslaat, met de tabs aan de onderkant (zoals in Excel). Op de iPhone moet het aantal tabs max 4-5 genomen worden, anders worden de tabs te klein om met de vinger te selecteren. Veel applicaties combineren de Master Detail view met tabs, zoals iTunes;
* Utility Application: deze heeft een voor- en achterkant (Main view en Information view), zoals de Weather applicatie, klik op de (i) en de pagina draait om waarbij instellingen tevoorschijn komen;
* Empty application: biedt geen standaard user interface, maar het hoogst noodzakelijke waarmee dan zelf de user interface ingevuld kan worden. Handig voor meer ervaren iOS programmeurs die meer willen dan de andere projecttypen bieden.

**4.2 Het Model-View-Controller MVC ontwerpmodel gebruiken**

Het basis principe achter MVC is dat objecten die data (Model) representeren worden gescheiden van objecten die de user-interface (View en Controller) bepalen. Verander je een Model object, dan mag dat geen invloed hebben op de user-interface objecten.

Voorbeelden van View objecten zijn buttons en sliders op het scherm, in feite alles wat zichtbaar is voor de gebruiker is opgeslagen in View objecten. Model objecten en View objecten praten tegen elkaar via Controller objecten. Elk object hoort volgens de MVC theorie in een van de drie categorien. In iOS is er geen strikt onderscheid tussen View en Controller objecten, er is nl. een ViewController klasse beschikbaar die beide omvat. Het onderscheid tussen beiden is toch te zien doordat Controller objecten in .h en .m files worden gedefinieerd en View objecten in .xib files.

De relatie tussen View en Controller is niet automatisch, deze moet in iOS expliciet uitgewerkt worden. Als voorbeeld een schermpje met een button en tekstvak. Druk je op de button dan verandert de tekst in het tekstvak. Button en tekstvak zijn beide View objecten. Het is in iOS echter niet zoals in andere platforms zo, dat als je op een View object klikt, je direct de uit te voeren acties kunt implementeren. Dit loopt in iOS via de Controller code. Druk je op de knop, dan wordt een action in de Controller geactiveerd. Deze action moet gedefinieerd worden in de Controller. Het tekstvak moet als een ‘outlet’ worden gedefinieerd in de Controller, om toegang tot de inhoud ervan mogelijk te maken. Dit geldt voor alle user interface elementen, sommige kunnen zowel een action als een outlet zijn.

**4.3 Basic interactie creeren**

In deze les wordt een iPhone app ontwikkeld waarin basale interactie wordt geimplementeerd. Creeer hiervoor een iPhone project in XCode (selecteer Single View Application), zorg dat alleen ARC is aangevinkt (Use storyboard en Include Unit Tests staan uit).

De aandacht gaat nu uit naar de 3 ViewController files die XCode heeft aangemaakt: met extensie .h (interface), .m (implementation) en .xib (user interface). User interface files hadden voorheen de extensie .nib, wat stond voor NextStep Interface Builder. De user interface files moesten toen met het aparte programma Interface Builder verwerkt worden. Sinds XCode 4 is Interface Builder in XCode geintegreerd en is daarmee tevens de extensie omgezet naar .xib.

We gaan nu de user interface voor het programma maken. Selecteer de ViewController.xib en open de Object Library (View – Utilities – Show Object Library). Drag achtereenvolgens een Label, een Text Field en een Round Rect Button in het user interface gebied. Maak de labeltekst leeg (het label is dan niet meer zichtbaar…) en zet een tekst in de button (bijv. ‘touch’).

We gaan nu de relaties tussen deze user interface elementen vormgeven. Als de button wordt ingedrukt moet een action uitgevoerd worden. Er moet ook wat gebeuren om de inhoud van het text field op te halen en de tekst in het label te wijzigen.

De code kan op twee manieren geschreven worden, op een langzame en op een snelle manier. In deze les beginnen we met de langzame manier, daarmee wordt ook duidelijk hoe de snelle manier op de achtergrond werkt.

We starten met de ViewController .h file. Hierin is de ViewController interface opgenomen, deze erft van de UIViewController klasse, die onderdeel uitmaakt van het UIKit framework. We gaan deze klasse uitbreiden met projectspecifieke code. Voeg een methode toe aan de interface. Deze is van het type IBAction, krijgt de naam changeLabel (= de selector) en heeft 1 parameter (id) sender. Het type IBAction (IB = Interface Builder) geeft de compiler aan dat de methode changeLabel aan een user interface element kan worden gekoppeld. IBAction is feitelijk hetzelfde als ‘void’.

De interface van changeLabel in ViewController.h is als volgt:

- (IBAction) changeLabel: (id)sender;

Klik nu op de Assistant Editor om de ViewController.m implementation file te tonen. Voer nu in de implementation sectie de code voor de changeLabel methode in. In de les is deze nog even leeg, enkel de methode definitie wordt opgenomen.

We hebben nu wat code, maar nog geen link met een user interface element gemaakt. Spring nu terug naar de ViewController .xib file. De koppeling tussen code en user interface element wordt gemaakt mbv de File’s Owner knop, de doorzichtige kubus midden in het XCode scherm. Deze representeert de connectie tussen de user interface (.xib file) en de code.

De link wordt als volgt gemaakt. Rechts-klik (of CTRL-klik) op de button, er verschijnt nu een vrij lang keuzemenu. Ga naar Touch Up Inside – dit komt overeen met het klik event, maar dat is geen keuzemogelijkheid in het menu – en klik op het kleine rondje naast de menu entry. Houdt de muisknop ingedrukt en sleep de lijn naar de File’s Owner knop. XCode laat nu alle IBAction methoden zien waarmee gekoppeld kan worden. In dit voorbeeld is er maar 1: changeLabel. Selecteer die en laat de muisknop los. De connectie tussen drukknop en methode is hiermee gemaakt.

De link van de knop kan later weer zichtbaar worden gemaakt door te rechts-klikken op de drukknop, of door te rechts-klikken op de File’s Owner knop of door de Connections Inspector te openen (ikoon met pijltje naar rechts, rechts bovenin XCode scherm).

Op dit moment gebeurt er nog niets zichtbaars als op de knop wordt gedrukt. We voegen nu een NSLog statement toe aan de implementatie van methode changeLabel in ViewController.m:

-(IBAction) changeLabel: (id)sender{

NSLog(@”Joepie de poepie”);

}

Dan build en run en voila, de string in het NSLog statement verschijnt in het output window als op de knop wordt gedrukt in de iOS simulator.

We hebben nog geen connectie gelegd tussen de code en het text field en het label. Dat gebeurt in de volgende les, waarbij de snelle methode van coderen wordt gevolgd.

**4.4 Snelle connecties creeren**

In deze les wordt op de snelle manier de link tussen user interface en code gelegd in XCode. Maak een nieuwe iOS Single View Application aan en creeer dezelfde user interface als in 4.3. Zorg dat de .xib file geselecteerd is en open de Assistant editor, zodat rechts de ViewController.h file in beeld verschijnt. De .h file is nl. de tegenpool van de .xib file.

Nu gaan we op de snelle manier de link leggen tussen de button en de code. Rechtsklik of CTRL-klik op de button en sleep vervolgens richting de interface section. Zodra dat gebied bereikt wordt verschijnt de melding ‘Insert outlet, action or Outlet Collection’. Dit verdwijnt direct als de interface section verlaten wordt. Laat de muisknop los en er verschijnt een window waarin de connectie gespecificeerd kan worden. Selecteer hier Action en vul changeLabel als naam in. Let ook op dat het event Touch Up Inside is geselecteerd, dit doet XCode automatisch voor buttons. Klik nu op Connect en de methode handtekening wordt in de interface section gezet. Het mooie is nu dat de implementatie van deze methode ook al in de implementation file ViewController.m wordt opgevoerd. Rechtsklik of CTRL-klik op de button en zie dat de connectie daar ook is gemaakt.

Doe nu het zelfde voor het textfield en het label – dus erop rechtsklikken en vervolgens slepen naar de interface sectie in de ViewController.h file.

Beide elementen zijn outlets. Noem het textfield ‘myTextField’, het type ervan is UITextField. In dit geval wordt een property declaration gecreeerd. Noem het label myLabel, verder hetzelfde als het textfield. Beide properties zijn van het type IBOutlet en ‘weak’, dit komt omdat de eigenaar van beiden de view is en niet de klasse (dan waren ze ‘strong’ geweest). Beide properties moeten synthesized worden, zodat getter en setter beschikbaar zijn. Dit is door Xcode al geregeld, zie de ViewController.m implementation file.

Het enige wat nu nog ontbreekt is de actie zelf, als de button wordt ingedrukt gebeurt er nu nog niets. De opzet is dat wanneer de button wordt ingedrukt in het label de tekst ‘Hello’ gevolgd door de tekst in het textfield verschijnt. Dit doen we door een NSString te creeren met een placeholder (voor de inhoud van het textfield):

NSString \*msg = [[NSString alloc] initWithFormat:@”Hello %@”,

[myTextField text] ]; // de ‘getter’ van myTextField

In het label moet nu de waarde van msg geschreven worden:

[myLabel setText: msg]; // de ‘setter’ van myLabel

Dankzij ARC hoeven we ons niet druk te maken om het releasen van objecten, dit regelt XCode automatisch.

Het programma kan gebuild en gerund worden. De iPhone simulator verschijnt weer. Wordt nu in het textfield geklikt dan verschijnt het keyboard automatisch. Het verdwijnt echter niet vanzelf, dat komt later.

**4.5 Het keyboard laten verdwijnen**

Wanneer in een textfield wordt geklikt verschijnt automatisch het keyboard, dit is standaard gedrag van UITextField objecten. Wat we ook verwachten is dat het keyboard verdwijnt als we klaar zijn met de input, dwz als we (1) de button loslaten, als we (2) op een gebied buiten het textfield klikken of als we (3) in het textfield op Enter drukken. Dat is echter niet het geval, dit gebeurt niet automatisch. Dus samenvattend hoeven we niets te doen om het keyboard te laten verschijnen, maar moeten we wel zelf het keyboard weer laten verdwijnen.

In deze les bouwen we oplossingen voor de eerste 2 gevallen. Het loslaten van de button is de eenvoudigste. Wanneer het keyboard verschijnt is het keyboard op dat moment het belangrijkste user interface element, in iOS wordt dat de ‘first responder’ genoemd. Zolang het keyboard de first responder status heeft, blijft het op de voorgrond. Door deze op te geven verdwijnt het naar de achtergrond. Dit doen we in de methode changeLabel uit 4.4 als volgt:

[myTextField resignFirstResponder];

Het weghalen van het keyboard bij het klikken buiten het textfield of de button vergt wat meer werk, zeg maar een truc. Dit gedrag wordt bereikt door een button te creeren dat het gehele scherm omvat. Het moet op de achtergrond zitten tov de andere user interface elementen. Wordt er dan op geklikt (en uiteraard niet op de andere button), dan moet het toetsenbord wegklappen.

Selecteer de .xib file en creeer een Round Rect Button. Geef deze de grootte van het volledige iPhone scherm. Het overlapt nu de andere user interface elementen. Verplaats het naar de achtergrond via menu opties Editor – Arrange – Send to Back. Het is nu een grote, witte button, niet wat we willen. Dat lossen we op met de Attribute Inspector, het 4e knopje uit de rij rechtsboven, juist onder de View icons. Zet het type op Custom, hiermee wordt het onzichtbaar.

NB De lijst met gecreeerde user interface elementen kan overigens opgevraagd worden door in het user interface scherm op het icon met driehoekje linksonder te klikken.

De nieuwe button moet nu nog aan de code gelinked worden. Rechtsklik op de nieuwe button, klik op Touch Up Inside en sleep vanaf het + teken naar de interface section. Het eigenschappen window verschijnt weer, vul als naam dismissKeyboard in en controleer dat de andere instellingen correct zijn. De methode dismissKeyboard wordt nu in de interface en implementatiefile toegevoegd. Voeg in de implementatie file het statement

[myTextField resignFirstResponder];

toe aan methode dismissKeyboard.

Het weg laten zakken van het keyboard bij het geven van Return werkt anders en komt in de volgende les aan bod.

**4.6 Het keyboard laten verdwijnen mbv delegation**

De events die tot nu toe zijn gebruikt om het keyboard te laten verdwijnen, vonden buiten het keyboard zelf plaats. Om het keyboard te laten verdwijnen door op de Return key te drukken moet gebruik worden gemaakt van ‘delegation’. Dit betekent eenvoudig gezegd dat een gebeurtenis op een user interface element plaatsvindt, het element dat zelf detecteert en een bericht daarover de wereld in stuurt. Dit bericht wordt vervolgens door een stuk code opgepikt dat dan een actie uitvoert.

Concreet in het onderhavige voorbeeld detecteert het textfield dat op de Return toets is gedrukt (=edit textfield is gereed). Dit ‘edit textfield gereed’ bericht kan aan een stuk code worden gekoppeld, dat vervolgens het toetsenbord inklapt.

Het realiseren in XCode hiervan gaat als volgt. Selecteer het textfield in de .xib user interface. Open de Utilities Inspector (ikoon met pijltje rechtsboven in XCode scherm). Dit toont diverse inmiddels enigszins bekende events, ook zichtbaar is waar eventuele outlets aan gekoppeld zijn. Bovenaan de lijst staat de delegate outlet. Hiermee kunnen we aangeven dat ‘een bepaald stuk code gedrag van het textfield voor zijn rekening neemt’.

Klik nu op het + teken rechts van delegate en sleep vervolgens naar het File’s Owner ikoon links van het user interface scherm. En dat is het. Er verschijnt geen pop-up window, zoals eerder bij het verbinden van user interface elementen.

Aan de hand van de documentatie wordt hier vervolgens enige uitleg over gegeven. Selecteer de Quick Help Inspector (ikoon rechtsboven met 2 golvende lijntjes) en klik op UITextField, het type behorende bij het textfield. Selecteer in het window dat verschijnt de UITextfield Class Reference. In de derde alinea onder Overview, valt te lezen dat een textfield object het gebruik van een delegate object ondersteunt tbv verwerken van editing gerelateerde notificaties. Er wordt verder verwezen naar het UITextFieldDelegateProtocol. Klik hier op.

Dit protocol definieert de berichten die verstuurd worden naar een textfield delegate als onderdeel van het editen van tekst. In de tekst die verschijnt wordt een opsomming gegeven van notificaties die het textfield kan melden (onder de kop Tasks). Voor ons van toepassing is de notification textFieldShouldReturn, deze vraagt de delegate of het textfield het indrukken van de Return knop moet verwerken. We kopieren nu de methode definitie

- (BOOL)textFieldShouldReturn:(UITextField \*)textField

in de implementation file ViewController.m, in de implementation section. Omdat we ons nu als delegate hebben geregistreerd, kunnen we op alle berichten die verzonden worden reageren, door de opgesomde methoden te implementeren (in de protocol reference is te zien lezen dat ze allemaal optioneel zijn, vandaar de keuze wel/niet implementeren).

In ons voorbeeld kunnen we de regel

[myTextField resignFirstResponder];

opnemen in de code van de methode, dit maakt het geschikt voor ons ene textfield. Echter heeft de methode een parameter textField, waardoor we een uniforme actie voor het indrukken van Return in alle textfields kunnen realiseren:

[textField resignFirstResponder];

De methode textFieldShouldReturn wordt geacht een waarde terug te geven die aangeeft of het bericht is afgehandeld. Dat is in ons geval zo dus voegen we

return YES;

nog toe en daarmee is de code gereed.

Het principe van delegation komt wellicht wat vreemd over, echter is dit een krachtig middel in iOS voor het afhandelen van user interface events. In de volgende les wordt uitgelegd wat hier achter zit.

**4.7 Het gebruik van delegation in iOS**

Delegation wordt zonder uitzondering in alle iOS apps toegepast. In andere talen wordt delegation gerealiseerd dmv overerving (inheritance) of event handling.

In iOS betekent delegation het overdragen van werk van het ene naar het andere object. De regels van iOS die gelden voor delegates zijn:

* Ken de regels: deze liggen vast in het Delegate protocol en zijn gedocumenteerd en vanuit XCode op te vragen. Een delegate protocol is een formele lijst van methode namen die geimplementeerd kunnen worden. Voorbeeld in par. 4.6 is de UITextFieldDelegate protocol reference;
* Zeg dat je een delegate kunt zijn: markeer je klassen als zodanig. Voorbeeld:

@interface ViewController: UIViewController <UITextfieldDelegate>

* Implementeer de gekozen methoden. Voorbeeld in 4.6 is de methode textFieldShouldReturn;
* Verkrijg het werk (?): verbind jouw delegate met het delegerende object. Voorbeeld is in 4.6 de link die gelegd wordt tussen het delegate outlet en de File’s Owner.

**4.8 Alert messages creeren**

Hoe bereik je in iOS dat een alert box verschijnt? Dit laten we zien aan de hand van een nieuw iOS – iPhone – Single View project, genaamd AlertMe. We gebruiken hiervoor niet de al behandelde technieken, maar een manier waarmee een alert box verschijnt, direct na het opstarten van een iOS app.

Het programma dat gecreeerd wordt door XCode bevat in de ViewController.m file diverse functions die te maken hebben met events die mbt de iOS app kunnen plaatsvinden. Dit zijn er vele, in het XCode reference manual kunnen deze opgezocht worden door te zoeken op UIViewController.

NB In de les zijn dit er veel meer dan in mijn programma dat op dezelfde manier is gecreeerd. Rara hoe kan dat?

Wij gaan code toevoegen aan de ViewDidAppear functie. Deze functie wordt uitgevoerd zodra een iOS app is opgestart. Er zijn veel meer voorbeelden van functies met het woord ‘view’ erin, zie hiervoor de programmer’s reference.

De code ziet er als volgt uit:

- (void) viewDidAppear: (BOOL)animated{

[super viewDidAppear:animated];

// onze code voegen we hier toe, zie verder

}

Dit is de standaard code die er zou moeten zijn. Voer eerst de call naar de parent class uit, zodat alle noodzakelijke acties worden uitgevoerd.

Een alert box is in iOS een object van het type UIAlertView. Deze creeren we en initialiseren we met de initWithTitle functie:

UIAlertView myAlert = [[UIAlertView alloc]

initWithTitle:@”Alert title”

message:@”Alert message”

delegate:nil // delegate gebruiken we niet

cancelButtonTitle:@”OK”

otherButtonTitles:<#(NSString \*), ...#>, nil]; // geen andere buttons!

De delegate zetten we op nil. Dit betekent dat de alert box een message naar een nil (niet bestaand) object zal sturen, maar dat is toegestaan in iOS en zal geen problemen geven met compileren, builden en runnen.

Voor de laatste parameter otherButtonTitles kunnen zoveel buttons opgegeven worden als gewenst, door de strings op te geven. De lijst moet altijd met nil eindigen. In de les worden geen andere buttons gebruikt dan de cancel button, dus voegen we hier enkel ‘nil’ in. Wil je met meerdere buttons werken, raadpleeg dan het reference manual

We hebben nu een alert box gecreeerd, maar die wordt nog niet getoond. Dat moeten we nog expliciet aangeven:

[myAlertView show];

Bij opstarten van de applicatie wordt de alert box getoond. Maar ook alleen na het opstarten! Wordt het programma met de Home button op de achtergrond geparkeerd en daarna weer opgeroepen, dan verschijnt geen alert box. Om dit te bereiken is meer kennis van iOS multi-tasking nodig.

**4.9 De iOS applicatie life cycle**

Hoe wordt een iOS applicatie gestart en uitgevoerd, wat gebeurt er als op het programma icon op de iPhone wordt geklikt?

Zoals alle Objective C programma’s, start een iOS app ook met main(). Deze zet XCode in staat in de Supporting Files folder van een project, omdat we er verder niets mee hoeven te doen.

Main() doet 1 aanroep: naar UIApplicationMain. Dit is een voorgedefinieerde functie in iOS die een UIApplication object creeert. Dit object representeert een standaard iOS programma, het zet een run-loop op zodat het programma actief blijft, het zorgt ervoor dat het op events reageert etc.

Onze iOS applicatie is geen standaard applicatie en we moeten aangeven hoe. Hiervoor wordt delegation (en niet inheritance) gebruikt in iOS. We moeten een klasse creeren die een delegate is van het standaard iOS UIApplication object. En dit doet XCode voor ons als we een iOS project creeren. In AppDelegate.m kunnen we dmv eigen code reageren op events op applicatie niveau, zoals het event ‘applicatie gestart’, dat door de methode didFinishLaunchingWithOptions van het UIApplication object wordt afgevangen.

Op dit punt hebben we nog geen user interface. Het eerste dat de functie didFinishLaunchingWithOptions moet doen, is melden dat een user interface nodig is. Dit wordt in twee stappen gedaan. Eerst creeert didFinishLaunchingWithOptions een UIWindow object. Dit is het window van de user interface, het bestaat zolang de applicatie blijft bestaan. Het window is echter leeg. In de tweede stap worden de view controllers geladen, deze bevatten de user interface controllers en bijbehorende code (.h, .m en .xib files). De voorbeeldprogramma’s die we tot nu toe hebben gemaakt bevatten slechts 1 view controller. Meer complexe programma’s maken gebruik van meerdere view controllers en wisselen geregeld van view controller.

Dit klinkt wellicht erg ingewikkeld. Er gebeurt ook behoorlijk wat, maar het meeste van wat besproken is wordt automatisch geregeld. Als programmeur heb je enkel te maken met de AppDelegate code en de ViewController code. De AppDelegate krijgt programma notifications (ook wel life cycle events genoemd) door, zoals didFinishLaunchingWithOptions waar je met eigen code op kunt reageren. De ViewController krijgt allerlei user interface notifications door, zoals viewDidLoad en viewWillAppear, daar kun je ook met eigen code op reageren. Wanneer de eerste view controller is geladen is het iOS programma volledig actief en komt de application notification ‘didBecomeActive’ door, daar kun je iets mee doen als je wil. Iets dergelijks gebeurt ook als op de Home button wordt gedrukt, dan komen applicationWillResign en applicationDidEnterBackground notifications door, waarover later meer.

**4.10 Multi-tasking in iOS**

Wordt een applicatie gestart dan wordt de didFinishLaunchingWithOptions notification gestuurd naar de AppDelegate van de applicatie. Drukt de gebruiker later op de Home button dan verdwijnt de app naar de achtergrond (ook dan worden notificaties naar de AppDelegate verstuurd). Wordt daarna weer op het app ikoon geklikt dan verschijnt het programma weer in zijn oude toestand op de voorgrond, echter worden dan andere notificaties dan didFinishLaunchingWithOptions verstuurd.

De volgende life cycle events zijn van toepassing:

* applicationDidFinishLaunchingWithOptions: eenmalig na opstarten programma;
* applicationWillResignActive: als bijv op de Home button wordt geklikt, kan meerdere malen voorkomen zolang het programma draait;
* applicationDidEnterBackground: idem;
* applicationWillEnterForeground: als de applicatie weer actief wordt gemaakt, kan meerdere malen voorkomen zolang het programma draait;
* applicationDidBecomeActive: idem;
* applicationWillTerminate: eenmalig als programma wordt afgesloten, bijv. door iOS als het beslist dat het geheugen nodig heeft of door de gebruiker.

Al deze methoden zijn opgenomen in AppDelegate.m. Voeg eigen code toe indien er iets moet gebeuren na een event. Hou er wel rekening mee dat dat snel – binnen een paar seconden – moet gebeuren, anders bestaat de kans dat iOS de app zonder pardon afsluit.

Een app die in de background staat doet standaard niets en neemt alleen wat geheugen in beslag. Er is echter een korte lijst van methoden die een app in de background mag uitvoeren (toestemming van iOS is nodig!):

* audio afspelen
* Reageren op locatie wijzigingen
* VoIP gebruiken, om gesprekken over internet te voeren
* Download Newsstand content (??)
* Verbinding houden en communiceren met een accessoire (bijv headset oid)

**Hoofdstuk 5 Troubleshooting**

**5.1 Een eenvoudige applicatie trouble shooten**

Het voorbeeldprogramma betreft het updaten van een label met de actuele tijd wanneer op een button wordt gedrukt. De user interface bestaat uit 2 labels, ‘time’ en een lege waarin de actuele tijd wordt weergegeven, en een ‘refresh’ button.

De methode updateTime zorgt voor het bijwerken van de tijd:

- (IBAction)updateTime:(id)sender {

NSDateFormatter \*dateFormatter = [[NSDateFormatter alloc] init];

[dateFormatter setDateFormat:@"HH:mm"];

NSString \*time = [dateFormatter stringFromDate:[NSDate date]];

[lblTime setText:time];

}

De code compileert en build prima. Maar wanneer op de ‘refresh’ button wordt gedrukt verschijnt geen tijd in het label veld. Dit is een veel voorkomende fout.

Onderzoek mbv de Utility Inspector de user interface elementen. Selecteer de Connections Inspector (ikoon rechtsboven met pijltje naar rechts). Selecteer nu de ‘refresh’ button. In de Sent Events is te zien dat het Touch Drag Outside event gekoppeld is aan updateTime via de File’s Owner. Echter is dit het verkeerde event, het moet nl. zijn Touch Up Inside.

Verbreek de verbinding tussen het Touch Drag Outside event en updateTime. Creeer een connectie tussen het Touch Up Inside event en updateTime. Dit kan op de gebruikelijke manier via de File’s Owner, maar ook kan mbv de Assistant Editor direct met de rechter muisknop de button aan updateTime gekoppeld worden.

Compileer en build opnieuw. Dit gaat prima, echter verschijnt nog steeds geen tijd in het label veld als op de ‘refresh’ button wordt gedrukt. Het kan zijn dat updateTime niet wordt aangeroepen. Om dat te controleren plaatsen we een NSLog statement in het begin van updateTime. Opnieuw builden en runnen en na een druk op de knop verschijnt de NSLog string in de output console.

We hebben gecontroleerd dat de actie (indrukken refresh button) gekoppeld is aan een methode. We hebben nog niet gecontroleerd of de outlet al is gekoppeld. Kijk in ViewController.h. Daar is het UILabel lblTime gedefinieerd. Links in de ‘gutter’ is te zien dat dit label nog niet is gekoppeld. Hier staat nl. een leeg cirkeltje, in tegenstelling tot bij updateTime waar het cirkeltje is gevuld. Klik op het cirkeltje en sleep het naar het label in het .xib window. Merk op dat alle code voor het label al beschikbaar was: de definitie in ViewController.h en de synthesize code in ViewController.n waren al beschikbaar. Was dat niet het geval, dan kan dit automatisch met XCode gecreeerd worden door in de .xib file op het label te rechtsklikken en het naar de interface sectie in VieController.h te slepen.

Opnieuw builden en runnen en nu werkt de code correct. Het niet koppelen van user interface elementen met code is een veel voorkomende beginnersfout.

**5.2 Breakpoints creeren en de XCode debugger gebruiken**

Tot nu toe hebben we mbv NSLog statements wat aan debugging gedaan. Naarmate de applicatie groter wordt is dat niet echt handig en kan beter gebruik worden gemaakt van de XCode debugger.

Het voorbeeld betreft een eenvoudige iPhone app met een ‘boom!’ button die de waarde van een label aanpast als er op wordt gedrukt. Zie voor de code het project DebugStarter in de Exercise Files folder.

De code is syntactisch correct, maar wordt het programma uitgevoerd, dan treedt een Division by 0 fout op: ‘Program received signal EXC\_ARITHMETIC”.

Links in het scherm is de Debug navigator zichtbaar. Deze toont de stack trace van het programma (wie roept wie aan). Onder aan dit window is een slider waarmee het detailniveau van de debug informatie kan worden geregeld. Voor de fout die nu is opgetreden helpt dit onvoldoende om de oorzaak te achterhalen.

Onder de code worden 2 windows weergegeven, links met waarden variabelen en rechts de console output. Let nog op dat gekozen kan worden uit lokale variabelen, alle variabelen en een auto view (die niet alles laat zien). De waarden van variabelen kunnen ook worden opgevraagd door met de muis over variabelen heen te bewegen.

In de code is zichtbaar wat het probleem is: variabele secondVal is 0. We stoppen nu het programma en zetten een breakpoint in de ‘gutter’ bij het statement waar firstVal wordt geinitialiseerd. Er verschijnt een blauwe pijl bij het statement. Klik er nog een keer en de blauwe pijl wordt transparant, om aan te geven dat het breakpoint gedeactiveerd is. Weer klikken activeert het breakpoint etc. Wil je ervan af dan sleep je het naar links, het verandert dan in een wolkje.

Laat het breakpoint staan en run het programma. Het stopt nu bij het breakpoint. Direct onder aan het code window bevinden zich een aantal debugger opties waarmee op verschillende manieren door de code kan worden gestapt. Klik op Step Into en we komen in methode calculateFirstValue terecht. Ga zo door tot we in methode calculateSecondValue terechtkomen. Klik hier op de Step Over button om over de instructie waarin calculateFirstValue wordt aangeroepen te springen. Creeer een breakpoint op het return statement van calculateSecondValue en klik op de Continue Program Execution button.

Variabele a wordt teruggeven maar heeft de waarde 0. Pas deze aan in het variabele window, zet hem op 300 en vervolg programma executie. De string ‘the result is …” verschijnt nu in de simulator, ten teken dat het deze keer goed is gegaan.

Als laatste tip: alle breakpoints kunnen met 1 druk op de knop worden gedeactiveerd door op de Breakpoints toggle knop boven in het XCode window te klikken.

**Hoofdstuk 6**

**6.1 User interface files creeren**

In deze les gaan we andere user interface elementen toepassen dan de buttons, textfields en labels die we tot nu toe geimplementeerd hebben. We creeren in XCode een nieuwe single view application en selecteren de .xib user interface file. In het ‘dock’ in het midden zien we 3 icons:

* File’s Owner, geeft de connectie tussen de .xib user interface en de view controller .m en .h files;
* FirstResponder, dit hebben we met een textfield gebruikt. Het is voor ons een placeholder die we verder niet gebruiken;
* View, dit is een UIView object. Klik er op en het gehele iPhone scherm wordt geselecteerd, dit is het scherm waar we mee bezig zijn.

In het iPhone scherm wordt de status bar met battery indicator getoond. Deze wordt altijd getoond en maakt geen deel uit van de App die we aan het programmeren zijn en wordt getoond om een zo realistisch mogelijk beeld van de eigen App op het iPhone scherm te geven.

Een standaard iPhone scherm is 320x480 pixels. Het retina display van de iPhone4 en 4S heeft een 2x zo grote resolutie. Wanneer de Size Inspector (direct rechts van de Attribute Inspector) wordt getoond, geeft deze aan dat de afmeting 320 bij 460 is. Dit komt vanwege de status bar, dus 20 pixels hoog is. De afmeting in de Size Inspector wordt in ‘points’ getoond, niet in pixels. In geval van een retina display is 1 point gelijk aan 2 pixels. Er zijn echter een aantal plekken in XCode waar onderscheid kan worden gemaakt tussen een standaard iPhone en een retina scherm, bijv. in de project settings, waar bij ‘Launch Images’ onderscheid wordt gemaakt tussen beide.

We gaan weer terug naar de Attributes Inspector. Hier kunnen allerlei scherminstellingen worden ingesteld, zoals scherm orientatie, achtergrondkleur, de (kleur van de) top bar, etc. Dit zit onder Simulated Metrics, het aanklikken van bijv. de top bar levert geen functionaliteit op, maar dient enkel om inzicht te krijgen in het uiterlijk van de user interface. Onder ‘View’ kan o.a. ook de achtergrondkleur worden ingesteld.

We zoomen nu in op de user interface elementen (Objects) en bekijken de Controls, dit zijn er overigens maar 10 (NB dit betreft de basis controls). We zetten er een paar op het scherm. Rechts in het XCode window worden de eigenschappen getoond van de geselecteerde control, daar kunnen ook weer diverse instellingen van de control worden ingesteld.

In de uitgebreide view van het Dock – klik daarvoor op het icon met het naar rechts wijzende pijltje links onderaan het user interface window – zijn de reeds geplaatste controls zichtbaar. Ze zijn ook zichtbaar bovenin de ‘Jump bar’, direct boven het user interface window.

We hebben nu diverse controls in de huidige view geplaatst. We kunnen ook nog een view binnen de huidige view maken, bijv. om een aantal controls te groeperen om ze als een eenheid te gebruiken. Zoek daarvoor op ‘View’ rechts in het XCode window of selecteer daar ‘Windows & Bars’. Er verschijnt dan een View element.

Om controls in een View te groeperen, selecteer een aantal controls in het user interface scherm (drag een rechthoek) en ga naar menu Editor – Embed In – View. De controls zijn nu opgenomen in een view, in het dock is goed te zien dat de controls gegroepeerd zijn.

Naast de elementaire controls die we in beeld hebben gehad, zijn er nog veel meer, complexere controls. De data views gaan we eerst bekijken.

**6.2 De picker control gebruiken en aanpassen**

Creeer een nieuw ‘single view’ project, noem het PickerControl en selecteer vervolgens het ViewController .xib window. Selecteer in het Object Library window rechts alle pickers (voer ‘picker’ in het zoekveld onderin het Object Library window). Selecteer hieruit de date picker en sleep dit naar het .xib window.

Er kunnen verschillende modi gekozen worden voor de date picker. Dit staat rechts bovenin de Date Picker attributen: zo kunnen tijd, datum, datum en tijd en de countdown timer geselecteerd worden. Wij gaan voor ‘datum’.

Je kunt nu al een succesvolle build en run doen om vanuit de date picker een datum-tijd in te voeren. We gaan nu een programma schrijven waarmee de dag van de week wordt getoond van de geselecteerde datum. Dit doen we aan de hand van een button. Als deze wordt ingedrukt wordt in een alert box de dag van de week getoond.

Selecteer een button in de object library en sleep dit naar het .xib window. Rechtsklik de button en sleep naar de interface sectie in ViewController.h om de definitie ervan toe te voegen (is van het type IBAction dat we al kennen). Noem de functie toonDatum (displayDay in de cursus).

Om de datePicker vanuit de code te adresseren – zodat de waarde kan worden opgevraagd - moet er een outlet voor worden toegevoegd. Rechtsklik op de date picker en zoek ‘referencing outlets’ op. Klik op het cirkeltje en sleep dit naar de interface sectie in ViewController.h, de outlet wordt dan automatisch toegevoegd.

We gaan nu de functie toonDatum implementeren. Deze staat in ViewController.m. Wanneer op de button wordt gedrukt, moeten we de datum die in de date picker is ingevuld opvragen, dit gaat als volgt:

NSDate ingevuldeDatum = [datePicker date];

Vervolgens creeren we een date formatter object, waarmee een datum in elk gewenst Unicode formaat kan worden opgeslagen ipv het interne datum formaat dat iOS hanteert. Google op ‘unicode date format’ en zoek binnen unicode.org naar het format dat de weekdagen representeert (4 E’s: “EEEE”).

De code om de dag van de week op te halen vanuit de datum in de date picker is als volgt:

NSDateFormatter \*formatter = [[NSDateFormatter alloc] init];

[formatter setDateFormat:@”EEEE”];

NSString weekDag = [formatter stringFromDate: ingevuldeDatum];

We maken nu nog code om dmv een alert box de dag van de week die hoort bij de ingevulde datum weer te geven:

NSString \*msg = [[NSString alloc] initWithFormat:@”Dat is een %@”, weekDag];

UIAlertView \*myAlert = [[UIAlertView alloc]

initWithTitle:@”Welke dag is dat?”

message: msg

delegate: nil

cancelButtonTitle: @”Dank u”

otherButtonTitles: nil];

[myAlert show];

**6.3 Het gebruik van data sources**