Software Developer's JOURNAL
new ideas & solutions for professional programmers

Vol.2 No.7 Monthly Issue 7/2013 (13) ISSN 1734-3933

iPhone development all you have to know

ADVANCE PAST MYTHS HURTLES AND THE INEXPLICIT OF IOS DEVELOPMENT

X-CODE 4 – YOUR FIRST PROJECT

IOS APP DEVELOPMENT METHODOLOGY – S2C2

EFFECTIVE USE OF BURN CHARTS FOR SCRUM TEAMS

BUILDING BIG DATA SOLUTIONS WITH THE BEST OPEN SOURCE HAS TO OFFER

Open source technologies MongoDB and Hadoop offer completely customizable and flexible platforms to meet your business' big data needs. Appnovation's big data solutions can process massive amounts of information and are built to be secure, stable and reliable in your IT enviroment. Appnovation delivers all our big data solutions in conjunction with our partners 10gen and Hortonworks.

In partnership with :

10gen

Hortonworks

Experts in Open Source Technologies

Creating Custom Solutions for Web, Mobile, Document Management, Big Data, Business Intelligence, E-commerce, Middleware and much much more!

Drupal · Alfresco · Sproutcore · MuleSoft · PhoneGap · Sencha · MongoDB · Hadoop

www.appnovation.com
ATLANTA · LONDON · VANCOUVER

APPNOVATION
TECHNOLOGIES

Dear Readers,

We would like to present you our brand new issue! This time we are willing to present all-in-one compendium of iPhone development.

As you know, this great discussion about Apple success is still alive. One may think that everything has been said and done, but that's obviously not true. There are still so many topics and questions to ask. Our team made its best to present you only the latest and most interesting news about iOS development. We hope that all those case studies and instructions shared by our specialists will help you improve your skills or get on the right track.

In this issue you will find two excellent articles written by co-workers Demetrios Kaligaris and Jacob Kennedy that have been working together on different projects for almost 20 years. They're presenting 'An Introduction to the Server' and 'iOS App Development Methodology – S2C2'. Furthermore, 'Integration Patterns for Native Mobile App Development' written by Equal Experts with great depth of topic and many useful instructions. Also, we have some new ideas for you, 'Everything you need to get started with GODPAPER' by Knight Zou or 'Build Better Apps by Leveraging the Power of BaaS' by Bobby Gill. Moreover, there are some useful and interesting topics for everyone like 'IOS Software Development With Adobe AIR' by Sosenyuk Oleg and 'Mobile App Revenue. Generating revenues from iOS mobile application' by Tridibesh Das.

To give you an insight into iOS development, Software Developer's Journal asked Doug Panchyshyn, well-know and high-specialized software developer some questions in an interview. Having said that, we would also like to share with all of you our excitement about up-coming issue, prepared specially by Doug. It will contain all information about iOS that you asked for and we're sure that it will be helpful for developers all over the world.

This time we also have some +EXTRAS for you. The extensive knowledge of Burn charts transferred very well and giving a good overview for the use of burn charts in 'Effective Use of Burn Charts for Scrum Teams' by Chris Merryman, 'Scaling to a Billion by Yaniv Pessach and useful tips in '6 Things to Consider When Partnering with a Mobile Developer' by AllianceTek.

We hope you'll find this issue useful and interesting and we're still working hard to improve for you!

Regards,
Anna Lakomy
Editor of SDJ Journal
and SDJ team

Software Developer's Journal
new ideas & solutions for professional programmers

team

Editor in Chief: Anna Lakomy
anna.lakomy@sdjournal.org

Editorial Advisory Board:
Laszlo Acs, Dawid Esterhuien

Special thanks to Raul Mesa

Special thanks to our Beta testers and Proofreaders who helped us with this issue. Our magazine would not exist without your assistance and expertise.

Publisher: Paweł Marciniak

Managing Director: Ewa Dudzic

Production Director: Andrzej Kuca
andrzej.kuca@sdjournal.org

Art. Director: Ireneusz Pogroszewski
ireneusz.pogroszewski@sdjournal.org

DTP: Ireneusz Pogroszewski

Marketing Director: Ewa Dudzic

Publisher: Hakin9 Media SK
02-676 Warsaw, Poland
Postepu 17D
Phone: 1 917 338 3631
http://en.sdjournal.org/

GETTING STARTED

A LITTLE BIT OF PRACTICE

EXTRA+

Advance Past Myths Hurtles

& the Inexplicit of IOS Development

For many new and experienced in iOS development there's a plethora of common things that can really slow some developers down. It can become overwhelmingly frustrating when we hit those hard walls of incomprehension. In a series of articles, I will identify and try to simplify many caveats that did impede a number of my students in their progress.

Approach: A broad range of items from simple to more complex will be covered over time. I will break out matters into topics of "Myths", "Hurtles", along with things we might perceive as "Inexplicit". I will attempt to simplify things as much as possible by using every day subjects the average person understands. As time proceeds I will produce and make available video tutorials on a unique new platform that will be announced in this publication and in the author's corner.

Myth: You must develop all of your code in Objective-C

The reality is, you can develop much of your code in straight C. You only need to learn how to convert or transition primitive C data into objects when you need to use Objective-C and vice versa (Listing 1). Many apps have more straight C code rather than Objective-C code in them, and there are many libraries that exist that are written only in C code with Objective-C wrappers.

Listing 1. *Sample Objective-C code living with C code*

```
// The following line is straight C code using
                the float primitive
float x, y;
// The following two lines convert Objective-C
                object data to a C float value
x = ([numberA.text floatValue]);
y = ([numberB.text floatValue]);
// The following line converts the data in x and
                y C code float value back to an
                Objective-C object
answer.text = [NSString
                stringWithFormat:@"%f",x+y];
```

Hurtle: Regular and Ongoing Beneficial Changes to Versions of Xcode

Apple has and will continue to make regular changes to their Xcode IDE (Integrated Development Environment). The main reason Xcode for iOS development continues to change is to make things easier and faster for the developer to develop iOS apps. Some developers have first chosen to take the time to learn how to develop iOS apps without using Interface Builder. I've noticed that these seem to advance quicker than those who first learn by the graphical approach using XIB's and Storyboards. On the other hand, there are many fine advantages to using Interface Builder. User Interface design is more naturally a visual task and more often quicker. It is a laborious effort to keep making manual changes to a user interface. Making adjustments to the values of CGRect to get it just right takes time. Also, using Interface Builder causes less code and less bugs to exist in your view controller making it easier for you to comprehend when you or someone else needs to make changes down the road.

The Inexplicit: SIGABRT a dreaded word and a common cause

I believe part of overcoming the fear of the dreaded Sigabrt is to not worry so much about all the information it produces, take your time, and look for the names of properties and other terms you are familiar with.

In the majority of cases I have found that the most common cause of a Sigabrt error my students encounter is the following:

1. They start by creating a single view app called TestApp using Storyboard.
2. They quickly drag two objects to their view, a label and a button.

3. Then they control-drag from their label object to attach it to their code in their .h interface file. Xcode can automatically create an Outlet property and they name this Outlet “myLabel”.
4. Then they go to the button and control-drag from this graphical object to attach the button to their code in their .h interface file, unwittingly create an Outlet property instead of an Action for the button and name this Outlet “test”.
5. Suddenly they realize that they should have created an Action instead of an Outlet.
6. Then, in their .h interface file, they delete the code that Xcode created to connect their button object as an Outlet.
7. A new control-drag is done for the button into their .h interface file, and this time they get it right by creating an Action that they name “test”.
8. They build and run their app and get an error message.
9. Clicking on the error message they notice in their Output Area a scary looking site.
10. Yikes and Yuk! (See Figure 1)

If we have a closer look at Figure 1, there is a hint on lines 4 and 5 where it says, 'this class is not key value coding-compliant for the key test.' When line 4, in our error, talks about 'this class', at first glance, it may seem to some that things are not very explicit. A new developer might ask, “What Class?”. If time is not taken, it may look like the class name is not being referenced in the error message. However, in fact, the name of the class that is having the problem is in Figure 1 on the first line crammed in between all kinds of other information.

I often recommend to my students to look for things they do understand and to take the time to read line by line for something familiar looking. You will note that on line 5 other information related to our problem, namely the name of the property 'test' and it is crammed in between the somewhat lengthy and cryptic looking error.

One suggestion I make to my newer students is to always remember the class name you make changes to. Do not make too many changes when first learning Xcode and Objective-C, but rather try to think back to the code or objects you changed. This should help you realize what caused the Sigabrt. In the above procedure, the Sigabrt is related to what they did in step 5 and 6. They realized they made a mistake by creating an outlet instead of an action and then deleted the code that Xcode generated to link their code to the UILabel object in their storyboard view. Xcode not only created the code inside the .h interface file but also updated the contents of the storyboard file. The problem is that if they delete the code that Xcode generated they must also delete something in the properties Xcode that changed.

One way to fix this is to open storyboard, click on the label and open the Connections Inspector (See Figure 2).

You will note at the bottom of the Connections Inspector that there is still a “Referencing Outlet” to “test” that should be removed. Clicking the small x showing “test” as still linking to the “View Controller” should solve the problem.

Myth Hurtle & Inexplicit: Delegation is a Myth to some, a Hurtle and a little Inexplicit

The myth about delegation is that you don’t really need it and you can just subclass. This simply is not true. It seems to be one of the most difficult things for new iOS developers to comprehend and so they keep trying to

```
1  2013-05-14 12:28:46.481 TestApp[1684:c07] *** Terminating
2  app due to uncaught exception 'NSUnknownKeyException',
3  reason: '[<ViewController 0x75acc70>
4  setValue:forUndefinedKey:]: this class is not key value
5  coding-compliant for the key test.'
6  *** First throw call stack:
7  (0x1c90012 0x10cde7e 0x1d18fb1 0xb79e41 0xafb5f8 0xafb0e7
8  0xb25b58 0x22f019 0x10e1663 0x1c8b45a 0x22db1c 0xf27e7
9  0xf2dc8 0xf2ff8 0xf3232 0x423d5 0x4276f 0x42905 0x4b917
10 0xf96c 0x1094b 0x21cb5 0x22beb 0x14698 0x1bebdf9
11 0x1bebad0 0x1c05bf5 0x1c05962 0x1c36bb6 0x1c35f44
12 0x1c35e1b 0x1017a 0x11ffc 0x1ded 0x1d15)
13 libc++abi.dylib: terminate called throwing an exception
14 (lldb)
```

Figure 1. *Yikes and Yuk*

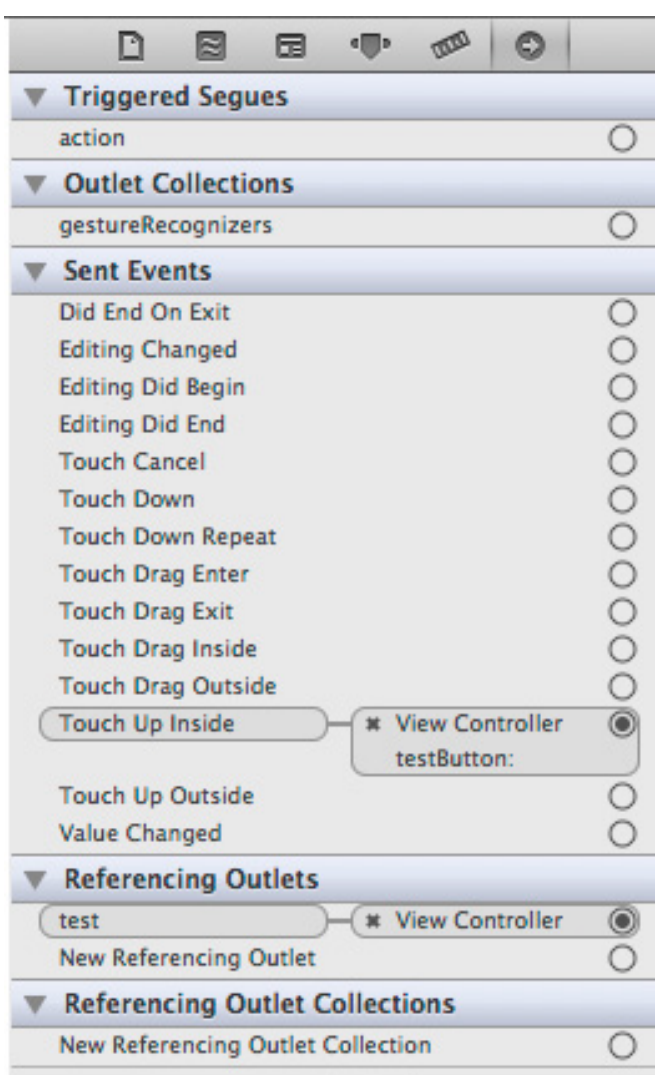


Figure 2. *Storyboard*

avoid it. However, it is one of the most important things to learn in order to advance in iOS development. IOS development code is riddled with it. You might think you are getting away without using it, but in fact, most don't realize they are using it when they use objects they drag onto their XIB's and Storyboards. There is more to learning about delegation and you cannot avoid learning more about it for very long. You probably will not advance very quickly if you do not learn about delegation in more detail.

Sometimes, I find that students are so anxious to learn about delegation that they simply do not slow down long enough to hear the reasons certain things are used. I also find that they need to slow down enough to read things in order to truly get the important points. Therefore, I am going to start by listing things out in point-like form. It may help you to slow down enough to really get it for it is not as complex as often perceived.

DELEGATION

A first step to comprehending delegation is to understand one of its main purposes.

If you think of the word delegation in the human sense, we delegate things because we can't do everything by ourselves; therefore, we pass on tasks to others who agree to handle them.

Also, as a developer, you want to avoid reinventing the wheel and instead reuse pre-written, proven, structured code whether that code is written by you, Apple, or someone else.

Using pre-written code usually aids you in programming faster and helps your programs to run faster. As a result, your code will probably be more stable and easier to maintain.

Apple makes some of their objects easy to "Reuse", by using a special "Design Pattern" called "Delegation" along with powerful graphical tools inside Xcode.

As an illustration in real life, when it comes to the word delegation, I like to use the example of an employer named "Secure Us" who hires an employee, named Bob, as a security guard.

Bob is a new employee who is nervous about being a security guard. However, "Secure Us" likes Bob and assures him that this is a very simple security guard job. He is handed a protocol card and told: "All you need to do is follow this list of protocols, some of which are required and some that are optional. Have a look at it as it is pretty straight forward. Let us know if you're ok with it."

PROTOCOLS

Required

1. Just before opening time, "Secure Us" listens for a timer to go off. When it does, someone from "Secure Us" will call you on your walkie-talkie and say: *Doors could open*. Handle this task by making sure you are at the end of the long corridor, around the corner from the door at your security desk, to get ready to press the button to open the doors.
2. From the outside of the building "Secure Us" screens all of the employees before they get anywhere near the door. Once they get to the door we will let you know on your walkie-talkie and say: *Doors should open*. Be ready to handle it and press the button to open the door to let them in.
3. Since you cannot physically see them from around the corner at your desk, we watch for them to go through the door once you open it. We will let you know on your walkie-talkie and say: *Door did open*" To handle this task, keep people moving by getting them to walk through the one way revolving door. As well, make sure that each person coming in signs the time sheet with their first name, last name, their time of arrival and signature. Check that their signature matches what you have on file for their name.

Optional

1. When an employee walks to your desk, you could greet them, but say no more than "Good Morning". If they try to start up a conversation you can simply explain that you are required to focus on handling the instructions that come from your employer at the moment.
2. Other optional task
3. Other optional task, etc.

Bob agrees to handle the behavioural tasks for all the required protocols that the Employer sets up for when certain door events happen.

In this sense, the Delegate is the employee "Bob" and the Delegator is his employer "Secure Us", who *listens and watches* for things that happen outside of the building. "Secure Us" lets Bob know when a *task* needs to be handled. "Secure Us" passes on or *delegates* the job to "Bob" who has agreed to handle the tasks.

Note, however, that the above scenario is a real life illustration. What we are trying to get across here is that Objects can have protocols or standards that need to be followed or agreed to by those on board to handle the tasks when different pre-conditions, events, and post-conditions take place. This is why, in Objective-C, we often see methods that contain the words, could, should and did within their method names. These are pre-conditions, events and post-conditions.

Inside Xcode there are "Delegating Objects". Some of these are represented by graphical images that Xcode lets you drag onto a Storyboard View. Other "Delegating Objects" can be non-graphical objects which you can

use. In fact, you've likely seen and already implemented many of these graphical representations many times. (See Figure 3).

The items in Figure 3 are just a few examples of some "Delegating Objects" from Apple. Once these are unarchived and instantiated as objects in your running code, they are able to pass tasks off to the classes you have written when your code gets instantiated into running objects.

These and other "Delegating Objects" can hand-over different types of responsibilities to your code. You make your class Objects agree to receive responsibility from a "Delegating Object". So, in this way, your code is a Delegate.

Somehow though, we need to tell the "Delegating Object", that whenever it hands out some responsibility for an event, that just took place, that it hand over this responsibility for that event to your code to handle what needs to be done.

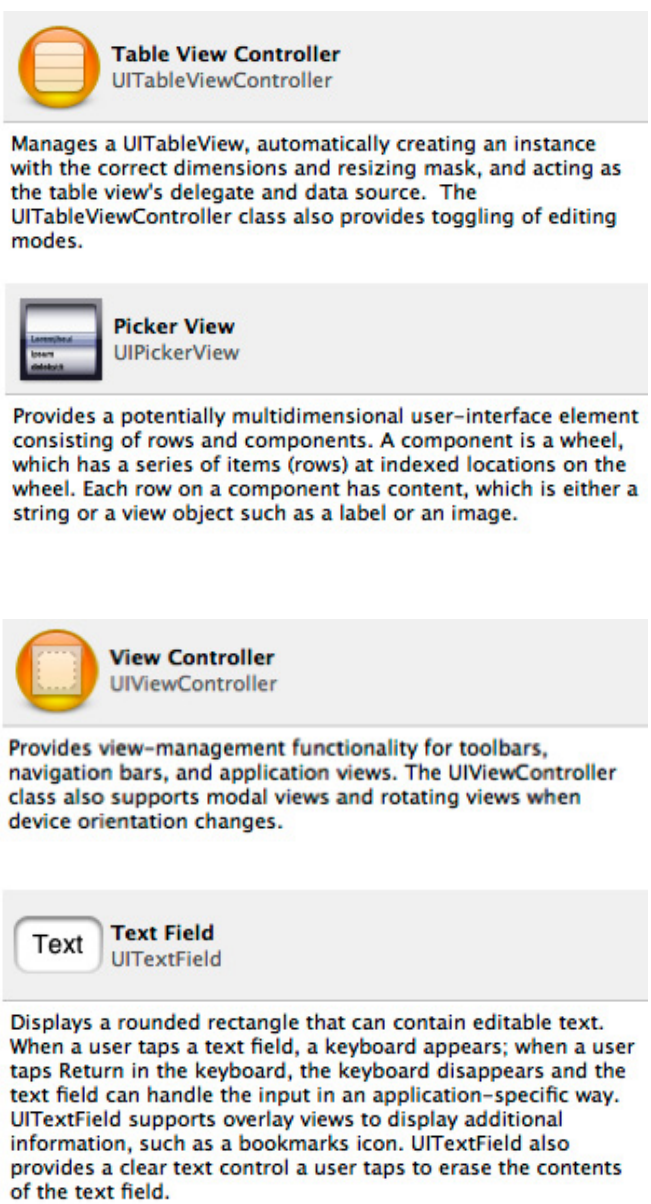


Figure 3. *Examples*

That is, our code needs to say: "Hey 'Delegating Object'", please hand responsibility over to my specific class that I wrote and not to some other classes in my program."

Xcode comes with a lot of functionality built into these Objects that Apple provides. Someone new to Objective-C, Cocoa, Xcode or iOS delegation, might ask the following question. "If I have to write the code to handle a certain event, why don't I just inherit all the functionality that I need into my own class from another class?"

Using inheritance for code-reuse is sort of like being forced to buy the whole car just to get a new set of tires. Or, like a bowl of shimmering jello, ends up entirely emptying out when you're trying to pour out just a bit. Inheritance has been used in all kinds of training material, and in example code in text books, to help new programmers understand how to use inheritance.

The reality is though that inheritance hierarchies often become very insubstantial since they cause an ever increasing number of irrelevant and unexpected behaviour that you never fully use in your code. With Delegation, on the other hand, you can reuse just a few very valuable functional methods. Don't get me wrong, inheritance has its place; however, we are focusing on Delegation here.

To finalize, delegates are used to extend the functionality of a class without the need to resort to subclassing. With a better understanding of the purpose of delegation, you should have a better idea of how it has something to do with an object listening for messages and then passing on the responsibility to another object. In a future articles, I will use what has been presented here in an actual example that include code. I will also produce and make available video tutorials using a unique new training platform that will be announced shortly in Software Development Journal and in their author's corner.

DOUG PANCHYSHYN

Doug Panchyshyn is an independent computer consultant who started developing in the C language back in the 80's and has launched into other languages and platforms with a focus on C. His current focus is on Objective-C for iOS and OS-X. He is also a UNIX/Linux web-server, database, and systems administrator. He has developed for both small and large organizations including many manufacturing companies designing and supporting ERP and process control, and has developed his own Relational Records Management solution for SoIntuitive.com. He develops for all major platforms while working for organizations such as IBM, AT&T Canada, Rogers Enterprises, Manulife Financial, Nortel Networks, the Ontario Government, the Canadian Institute for Health Information and others.

Xcode 4
Your First Project

Apple has revolutionized the software world, by providing a new way of selling apps. It minimizes the act of piracy and has created a one stop market place for all the apps developed by different individuals around the globe.

Thus creating a pool of innovative apps in a single bowl where every citizen will have an opportunity go at it. Apple's success has given the option to other software players like Microsoft, Google, Intel, Nokia etc, to follow the foot path of Apple only to shrink the globe to a village. This option has opened the door for many individual developers to convert their ideas and sell at the appropriate App Stores. However to get into next or intermediate level programming you need to be strong in concepts like MVC, classes, objects and general programming concepts. Knowing Objective C will be an advantage. Knowledge in C/C++ or any other object oriented language will make things easier to understand.

If you are new to programming not only for this level but even for the next level, with your good English knowledge, you can learn and implement things, when you are really open and happy to face challenges and have sheer perseverance and belief.

Requirements

iOS App Development requires a set of hardware & software requirements.

Hardware requirement

- Mac System (Cost Effective solution is Mac Mini)
- Mac Keyboard & Mouse. Mac Mini can use monitor / keyboard / mouse of the PC
- iPod/iPhone Device (To test the developed app). But, for this article we will be using the "Simulator" option available in Xcode.

Software requirements

Xcode 4 is a complete IDE (Integrated Development Environment) for building iOS apps. It comes with different built-in software supported to build iOS applications. The different software is listed below.

Interface Builder

Interface Builder helps create the front end design. It has lot of iOS based components called as views. Here are few of the views provided by Interface Builder

- Views
- Image
- Text box
- Label
- Button
- Picker (to choose different options)
- Tool Bar
- Navigation Bar
- Table View

Apple LLVM Compiler

Xcode 4 also includes Apple LLVM that acts as a compiler. The LLVM also supports Syntax highlighting, code auto-complete options. LLVM works in the background and follows each and every text we key-in and helps us to correct the mistakes in case of any incomplete or incorrect code. And, it doesn't just report the errors, at times it is intelligent enough to auto-correct the errors.

Version Editor

Version editor acts as a source code management option in Xcode 4. It helps you to compare two versions of your code side by side. It logs everything and helps keep track of previous check-ins.

Xcode 4 Debugger

Xcode 4 also has a debugger called LLDB.

Simulator
Simulator is similar to the actual device and is used to run output in Mac itself. Both iPhone & iPad Simulators are available in Xcode 4 and also with different iOS versions.

Architecture
Xcode 4 follows the MVC (Model-View-Controller) Pattern. A quick run-though on MVC:

- Model: mainly consists of data
- View: represents the output
- Controller: acts as a mediator between Model & View.

The detailed explanation of the above will be helpful at the next level of programming.

First App in Xcode 4

To start your project in Xcode, follow the steps provided below:

Step I
Open Xcode, which by default is located inside the "Application" folder. If it is not present in the "Application" folder, search for Xcode in the Mac Search option located at the top right area.

Figure 1. *XCode Welcome Screen*

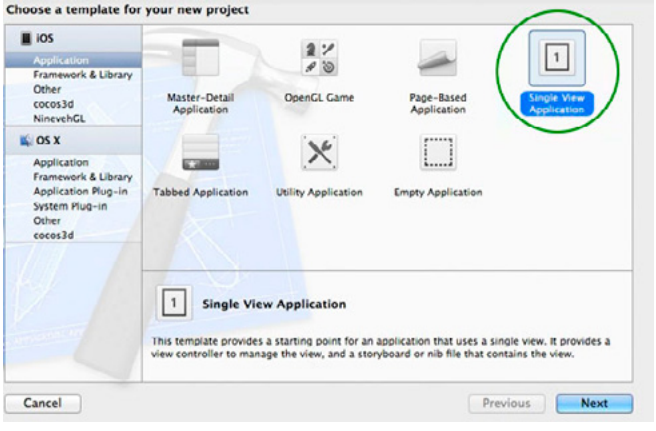


Figure 2. *Project Template Screen*

Step II
You will find a Welcome Screen with different options. Click "Create a new Xcode project" option (Figure 1). You also have another option to create a new project, which can be achieved through Xcode menu, From the Xcode menu, select File → New → Project.

Step III
A next screen will come with the different project templates, which can be selected to cater to different needs.

Note
The template selection screen contains options to create iOS apps/frameworks & Mac apps apps/frameworks/plug-ins etc.

For our project, select "Application" under iOS platform and then select "Single View Application" and click the "Next" button (Figure 2).

Step IV
Provide the project related details such is Product Name, Bundle Identifier, Class Prefix etc. We can name the project "Hello World".

Note
Bundle Identifier & Class Prefix can be learned in detail at the next level.

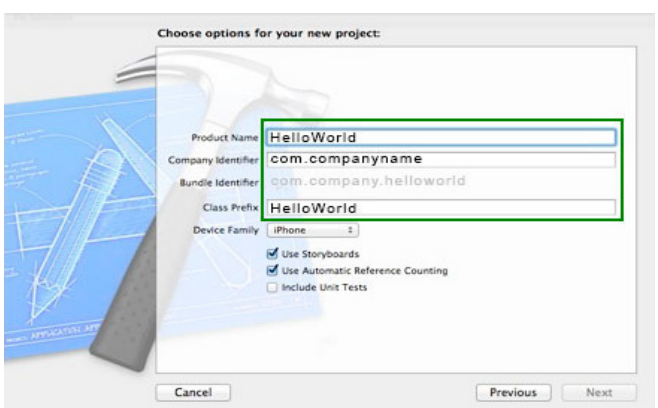


Figure 3. *Project Details Entry Screen*

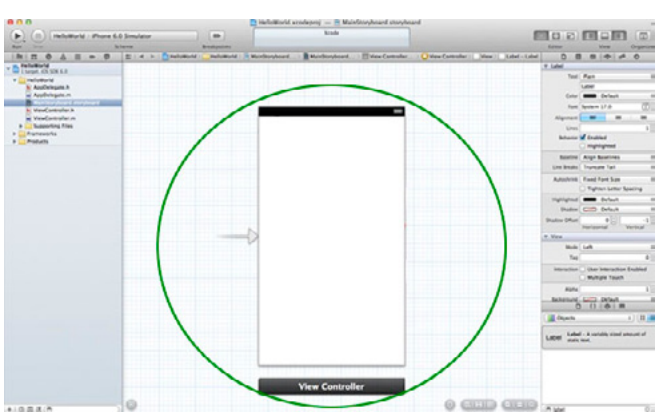


Figure 4. *Blank Storyboard Screen*

Provide the inputs as seen in the Figure 3.

Step V

Click on the *MainStoryBoard.storyboard* file from the left file list panel to get the following Figure 4.

Step VI

Drag the Label component from component List and drop it in the view as seen in the Figure 5.

Step VII

Double click on the Label component and key-in the following text "My Hello World App".

The text will be reflected in Text option of the Label component as shown in the Figure 6.

Step VIII

We are done with our first project. Before running our first app in the iPhone simulator we need to make sure we choose iPhone simulator from the Scheme menu (Figure 7).

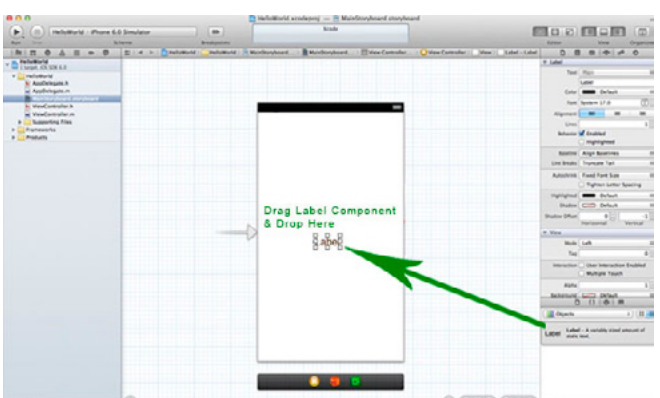


Figure 5. *Label Component Add Screen*

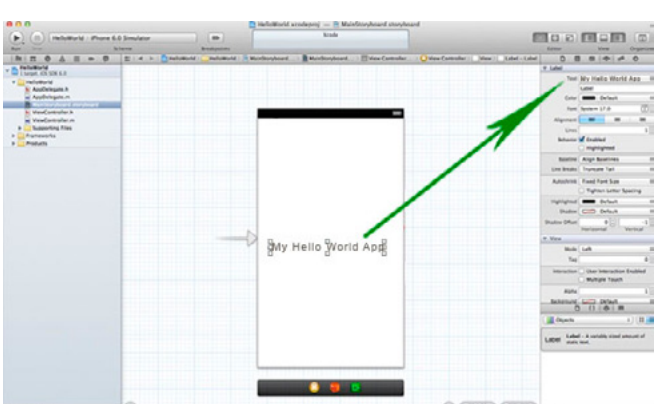


Figure 6. *Label Add Text Option Screen*

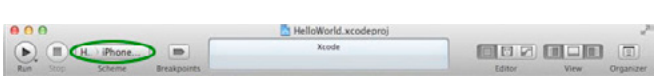


Figure 7. *Scheme Option Screen*

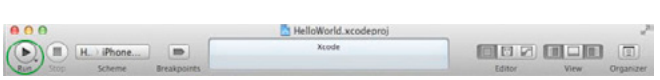


Figure 8. *Run Option Screen*

Glossary

- LLVM: A compiler that builds the app.
- LLDB: The Xcode debugger
- MVC: Model-View-Controller

On the Web

The best iOS developer resources are found here:

- *https://developer.apple.com/devcenter/ios/index.action*

The author with his team has developed many apps and have listed them under:

- *http://www.anandtechmedia.com*
- *http://www.appwings.com*

Step IX

Now its time to run and see our first ever app in the iPhone simulator. There are three methods to run our app:

- From Xcode menu select Product → Run (or)
- Press Command + R from Mac keyboard or Windows + R from PC keyboard (or)
- Click the "Run" button, which is present at the top left of the Xcode software as shown in Figure 8.

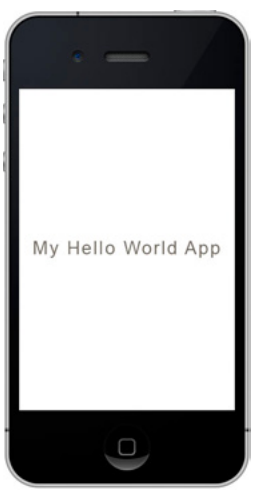


Figure 9. *Simulator Screen*

Hurray! Your first ever app runs in the Simulator like a charm as seen in the following Figure 9.

Summary

From this article you have learned about the software & hardware requirements needed to create an iOS app. You used Xcode to create a new Single View Application. You have learned how to add a Label component and run the app in iPhone Simulator.

SARAVANAN K (SARAN)

Saravanan K (Saran), CEO of "Anand Tech Media TM" (www.anandtechmedia.com), a smart phone development company that has been over the years developing hundreds of apps in iPhone & Android under its Brand "App Wings TM" (www.appwings.com).

Saran has over 14 years of experience in software & web development. He has started his smart phone development with "Anand Tech Media TM" since its inception from the year 2008.

Contact Saran at, saran@atmedia.in.

HPW EXECUTIVE SEARCH

Crazy but Smart Mobile Developer Geeks

& did you start programming when you were 10?
& are you super smart en innovative?
& are you someone who builds things to perfection?
& is your development work disruptive for society (in a good way)?
& do you have an unyielding passion for development?
& are you fun and social?
& are you reliable and trustworthy?
& do you like a beer?
& do you want to meet the office dog Snoopy?
& are you committed and willing to put the hard work in (and get rewarded for it…)?

then you need to meet this bunch of people…. and Snoopy!

Our client is developing a cross platform social experience where developers can utilize their value-adding tools and publish only quality apps in a clutter-free 'blue ocean' environment. It is our client's goal is to create a social platform were users will socially connect with their friends and family on the platform that will be playable across many different devices – be it a smart phone, tablet, desktop, or even a TV.

Our client has big plans to attract outstanding developers – both big and small – to its social platform. Her developers share common goals and visions for the company, but hail from all walks of life and speak dozens of languages, reflecting the global audience they serve.

Your Profile Our client is looking for experienced developers with in-depth knowledge of Mobile development with iOS or Android. If you have a proficiency in multiple programming languages and you have a proven track record of personal projects, please apply via info@hpwexecutivesearch.com to find out more!

About HPW Executive Search

HPW Executive Search is a Dutch owned executive search firm specialized in highly qualified ICT staff recruitment in the Netherlands. Due to our past experience in working at and on behalf of ICT organizations we are able to make successful matches between the organization, job requirements and the ambition of our candidates; whom we are mainly recruiting from our own network. We provide our services either from our offices in The Hague and Amsterdam.

Job searches carried out by our firm with the following specializations:

- Solutions, Hosting, Cloud
- ICT developer/ architect: Java, Scala, Python, PHP, C#
- Mobile development: Android, iOS
- Big Data, Business Intelligence (BI), Customer Intelligence (CI)
- Content Management Systems: Tridion, Drupal, Hippo, TeamSite
- Identity & Access Management (IAM) Consultants / Business Analists
- Agile / SCRUM Master
- (Technical) Team lead / ICT Manager

For our latest job opportunities please visit our website: www.hpwexecutivesearch.com

HPW Executive Search
Keizersgracht 62 - 1015 CS Amsterdam - the Netherlands - +31 (0) 20-520 7599 (Phone)
Koninginnegracht 19 - 2514 AB Den Haag - the Netherlands - +31 (0) 70-217 0001 (Phone)

An Introduction to the Server

The purpose of this article is to introduce you to a new tool for building systems. It has been designed to play nicely with the tools you already like to use, and eliminates common annoyances that are the less enjoyable parts of programming. Its goal is to let you do what you want do faster than you could without it.

The Server is the product of many years of hard work by a small team of very committed individuals who are passionate about software development. It strives to apply many of the lessons learned from building lots of commercial systems, and provide a key set of services to build systems rapidly.

Fundamentally, the Server offers these capabilities:

- Define data structures and relationships
- Create, retrieve, update, delete, undelete, and relate data, with complete version history
- Secure and share data
- Create and manage users and groups
- Communicate with any client using an easy to learn HTTP interface
- Convert an iOS device into a fully functioning application server

The Server has just been released on the iTunes App Store. It will turn your iOS device into a fully featured wirelessly accessible data modeling and multi-user access platform for you to use to build systems with that are centered around other devices using your device as their server. It breathes new life into anything that runs iO5 or better.

You'll need to provide your own wi-fi network, the development tools of your choice, and some initiative to try out something all new, and tread some new ground for software design.

Start it up and use a browser to connect to your device, and you'll find the API for the HTTP-Get interface is documented there, with one click access to try out each command.

The Server app provides built in user and group management, as well as real-time log file viewing. If you'd like to secure the transport layer, you can enable HTTPS.

When the Server is running on your device, any software development tool that has an HTTP client on your network can connect to it. It can handle lots of connections, so you can use it to share data any which way you can dream up.

Why You'll Want To Use It

The Server is a brand new way of building systems, built by experienced programmers to be a better way to store, secure, and access data. It provides a set of commands that can be called by any development tool that has an HTTP client. The purpose of this server's command and operating framework is to allow developers to focus on design and user interfaces, to more effectively build and operate systems.

It exposes a set of services over HTTP or HTTPS that allow a programmer to define, create, and manage data. It takes care of storing and securing data, so you can get to the business of designing your system, and counting on the server to store and give you what you want when you want it.

The design approach to creating and accessing data in this system is very powerful, but will likely take some getting used to. Everything is version controlled, and secured. Logical concepts like databases, tables, and rows are represented by defining, instantiating, and relating things together.

The Server is kind of like a NoSQL system, it's kind of like a database, it's kind of like an application server, it's kind of like a web server. While it is all of those things, it really is something you've never seen before.

You can think of it as a database because it lets you define the structures you want to use, and how the

structures should relate. Then it lets you bend those rules easily if you need to, without large redesigns or change scripts.

You can think of it as a data warehouse because it never forgets anything it was told to keep. If you want to see how something looked at a particular point in time, it is simply a few read requests away. This opens the door to some very interesting auditing possibilities.

You can think of it as a universal database because it works with any client because it doesn't need extra client access software.

You can think of it as a NoSQL system because communicating with it is kind of like communicating with name and value pairs, but it's more than that.

You can think of it as an app server because it hosts your system, and takes care of user permissions and access controls.

The engine driving this software is built entirely in C++ by a very small team, with a desire to build systems a better way. We think we've accomplished our goal. We hope that you try it, and agree.

Server's Data Structures

Conceptually, the server is based on Types and Entities. A Type is the definition of data, and an Entity is an instance of data.

A Type can express a fundamental data type, such as a string or a number, in which case it is called a Data Type. Alternately, a Type can express group of fundamentals, in which case it is an Entity Type. A Type can also express a database, in which case it is a Database Type.

Entity Types are joined with data Types through Type Components. Type Components relate an Entity Type to a Data Type. By doing this, a construct similar to a table with fields is created.

Entity Types are joined with other Entity Types through Type Relations. Type Relations relate an Entity Type to another Entity type. By doing this, a construct similar to a foreign key on a table is created.

An Entity is an instance of an Entity Type. The creation of an Entity can be though of as the presence of a blank record, as it does not yet have any data, although it could be created with a sequence to retain an order.

Once an Entity has been created, Entity Components can be created on it, to represent the data for the Entity. This is similar in concept to providing values for the fields on a table.

An Entity can be related with another Entity with an Entity Relation record. This is similar in concept to providing a key to another table into a foreign key field on a row.

The Server's seven major structures are:

- Component Type – The definition of fundamentals
- Type – Definition of Database, Entity and Data types
- Type Component – A link between an Entity Type and a Data Type
- Type Relation – A link between an Entity Type and an Entity Type
- Entity – An instance of an Entity or Database Type
- Entity Component – An instance of a Data Type
- Entity Relation – A relationship between two Entities

Component Type

Component types are the raw building blocks of systems. The basic component types are provided by the server, and these allow you to construct objects, and rules for objects to represent structures similar to tables and rows, and define the relationships and permissions to each of these.

Component types are extended with 'Types'. Each 'Type' that is defined has a component type that indicates its fundamental data type. This allows for a the creation of types to express all things needed for building a system. This could be a basic type such as a number, date, or string, or it could be an 'Entity' type to group these basic things together. It could also be a database type to group entities together. Any Component Type created by a user of the server is considered a 'Database' component type. For the most part, component types are read, and not added to. The Component Type server provides tools to read and create.

When you are building an application server up for the first time, one of the initial configuration steps is to create a component type to represent your database.

You'll then create a type to represent your database, and an entity of that type to bring it to life. Then, when you create entities to express your data, you'll associate each data entity with a database entity.

This way, everything is kept together, and you can have multiple databases of the same 'Type', each with its own unique data, in the same server.

Type

Types define the data types that can be used in the System. Each type has a 'Component Type', that indicates the type of data it represents.

Types define the data that system can manage. This includes entities, and entity components, which are created to represent information in the server. Entities and entity components can be thought of as instances of 'Types'.

A Type that is of Component Type 'Entity' can be thought of as a table definition. Instances of Entities that are created with the 'Entity' Type can be thought of as rows in the table defined by the 'Entity' type.

A Type that is of a 'Data' Component Type (which means it is not of type 'Entity', and not of a user created 'Database' Component Type) can be thought of as a field for a table.

Note that the type can be shared across many tables. For example, a String type named 'First Name' can be put onto a 'Customer' type and also a 'Vendor' type. This construct allows data searches by field type across table types if desired.

Type Component

Type Components relate an Entity Type to a Data Type.

This allows for defined data types to exist on entities. For example, an Entity Type called Contact could get a String type called First Name by creating a Type Component with both Type Guids.

Note that this construct is for defining a standard model, but the absence of a Type Component does not stop the creation of data on entities. This means that you can extend the fields a logical table on an individual row level.

Type Relation

Type Relations relate an Entity Type to another Entity Type. This allows for defined relationships to exist between entities. For example, an Entity Type called Contact could get a related Entity Type called Address by creating a Type Relation with both Type Guids.

Note that this construct is for defining a standard model, but the absence of a Type Relation does not stop the creation of relationships between entities. This means that you can create relationships on an individual row level.

Listing 1. *A UserCredentials object in result container expressed in JSON*

```
{
   "M":"UC",
   "R":
   {
      "UG":"01234567-89AB-CDEF-FEDC-BA9876543210",
      "NM":"SuperUser",
      "PW":null,
      "A":true,
      "RS":255,
      "WS":255,
      "CT":true,
      "CE":true,
      "UP":true,
      "IUP":false,
      "GG":
      [
"01234567-89AB-CDEF-FEDC-BA9876ABCDEF"
      ]
   }
}
```

Listing 2. *A Type Object in result container expressed in JSON*

```
{
   "M":"EL",
   "SC":"-1",
   "ST":"Type operation was successful.",
   "R":
   {
      "List":
      [
         {
            "Type":
            {
               "TG":"01234567-89AB-
                  CDEF-0000-000000005678",
               "CTG":"01234567-89AB-
                  CDEF-0000-000000000000",
               "TNM":"Example Entity Type",
               "TNT":"Example Entity Type Notes",
               "RMT":false,
               "CA":
               {
                  "O":"01234567-89AB-CDEF-
                  FEDC-BA9876543210",
                  "OT":0,
                  "R":"01234567-89AB-CDEF-
                  FEDC-BA9876543210",
                  "RT":0,
                  "W":"01234567-89AB-CDEF-
                  FEDC-BA9876543210",
                  "WT":0,
                  "RS":0,
                  "WS":0,
                  "AT":"2013-04-26 00:21:12",
                  "AU":"01234567-89AB-CDEF-
                  FEDC-BA9876543210",
                  "CT":"2013-04-26 00:21:12",
                  "CU":"01234567-89AB-CDEF-
                  FEDC-BA9876543210",
                  "D":false,
                  "C":true,
                  "V":1
               }
            }
         }
      ]
   }
}
```

Entity

Entities are instances of 'Entity' or 'Database' Types.

An entity that is created with an Entity Type can be thought of as a row in a table. The creation of the entity record creates the row, but not any data. The data for the row is added using Entity Component commands.

An entity that is created with a Database Type can be thought of an instance of a database. The 'Database' entity is a container for 'Entity' type entities.

Entity Relation

Entity Relations relate an Entity to another Entity.

For example, an Entity of type Contact could get a related Entity of type address by creating an Entity Relation with both Entities.

There does not need to be a TypeRelation in place for the Entity Types for this relation to be created, but if there is the system will validate the Min and Max partners rules from the Type Relation.

Entity Component

Entity Components are data attributes on an entity. For example, an Entity of type Contact could get a data Entity of type First Name by creating an Entity Component with Type 'FirstName' on an entity of type 'Contact'.

There does not need to be a TypeComponent in place between the Entity Type and the Data Type for this EntityComponent to be created, but if there is the system will validate the Min and Max instances, and uniqueness rules from the Type Component.

The EntityComponent Command can be used to access all data, because it checks to see which data type specific server needs to be used. Alternately, the data type specific servers can be used directly.

The Entity Component commands will determine the correct data type specific server to use by checking the Component Type of the TypeGuid, or by scanning each data type server for the presence of the provided EntityComponentGuid.

Listing 3. *A PHP example showing the GetUserCredentials command invoked on an iPad*

```
<?php

// set server host/credentials
$host="http://jacob-kennedys-ipad-v1.local:1125"; //
                    Host name

// username and password sent from form
$formUsername=$_POST['username'];
$formPassword=$_POST['password'];

// build & encode request string to validate the user
$requestString = $host . '/GetUserCredentials?UN=' .
                    urlencode($formUsername) . '&PW='
                    . urlencode($formPassword) .
                    '&output=JSON&BS=' . time();

// execute request to the iPad server
$requestResponse = file_get_contents($requestString);

if ($requestResponse == FALSE)
{
   echo('Error received from file get contents');
}

// parse json response
$results = json_decode($requestResponse);

// Get the value from the Mode (M) name value pair.
                    We want to see if it contains a
                    value of  'UC'
$resultMode = $results->{'M'};

if ($resultMode == 'UC')
{
   // now that we know we have a UserCredentials
                    object, pull it out of the Result
                    name value pair
   $userCredentials = $results->{'R'};

   $userGuid = $userCredentials->{'UG'};
   $active = $userCredentials->{'A'};

   echo('Active: ' . $active);
   echo('UserGuid: ' . $userGuid);

   if ($active == true && $userGuid != null)
   {
      //valid user - load vendor summary/home page
                    passing UserGuid
      echo('Handle successful login');
   }
   else
   {
      //invalid user - drive back to original page
                    with error
      echo('Handle Failed Login');
   }
}
else
{
   echo('Report Error');
}
?>
```

Server's Command Structure

The server's external interface groups commands into logical sub servers, named after the data structures that they manage. The commands available in each of these data servers are consistent one to another. The commands that each of these servers provide are:

```
Create, Retrieve, Update, Delete, Undelete
```

In addition to the sub servers that manage the data structures, there are two other servers that round out the functionality. These are the 'Query' server, and the 'Root' server commands.

The Query server provides a set of commands to ask the Server for information about the data it has. Each query command has a set of optional parameters that can be used to filter the data returned. The query server commands return table structures containing the keys that can be used to read the data that the query matched.

The Root Server commands provide utility commands to perform tasks such as server connection and disconnection, user and group inquiry, and transaction management.

Using the HTTP-GET interface, a single command is executed with each request to the server. The results of the command's execution are returned in the HTTP response, and will be expressed as a Result object, serialized in XML or in JSON.

It is important to note that by using the HTTP-POST interface, a command batch can be submitted to the server that will succeed or fail as a whole, within a transaction. Using the HTTP-POST interface requires that the HTTP message body contain an XML or JSON serialized Command or Command Batch.

Example Results

A result object containing a UserCredentials object: Listing 1. A result object containing a Type object: Listing 2.

Example Server Interaction

This code example demonstrate the interaction between an Apache Web server, and an iPad server.

In this example, login credentials provided to the PHP server are used to retrieve a UserCredentials object from the server, using a Root Level command called 'GetUserCredentials. The object returned form the server is checked to make sure it is Active, and has a Guid value that is not null to confirm that the user has access to the system. The users that are allowed access to the Server are managed from the iOS application (Listing 3).

Summary

The Server is a tool that lets Software Developers build systems in a new way, using any tool with an HTTP client. The Server takes care of the mechanicals of the commonly repeated parts of almost all business systems, and lets the Developer get to the business of developing.

It brings a brand new use to any iOS 5+ device, and allows it to be used in a way that is brand new. Building a system with an interactive server with a touch screen interface, with immediate access to all the relevant information is something that really needs to be tried. We're confident that you're going to enjoy it.

DEMETRIOS KALLERGIS

Demetrios Kallergis is a Software Developer, and founder of Kallergis Consulting Limited who is passionate about his work. He's been working in software development for 15 years, in a variety of industries and environments that have helped shape his view on software development, and have shaped his desire to make things better.

You can reach Demetree at demetree@kallergisconsulting.com.

Demetree and Jacob Kennedy, the author of SDKToday have been taking on interesting projects for almost 20 years, and have always found that working together has produced excellent results. The Server is the latest example of that collaboration. Many thanks go to Jake for his design reviews and accommodating ears over the lengthy design and build process of the product.

Jake and Demetree are building the next generation of systems with the Server as the foundation.

ACUNU ANALYTICS

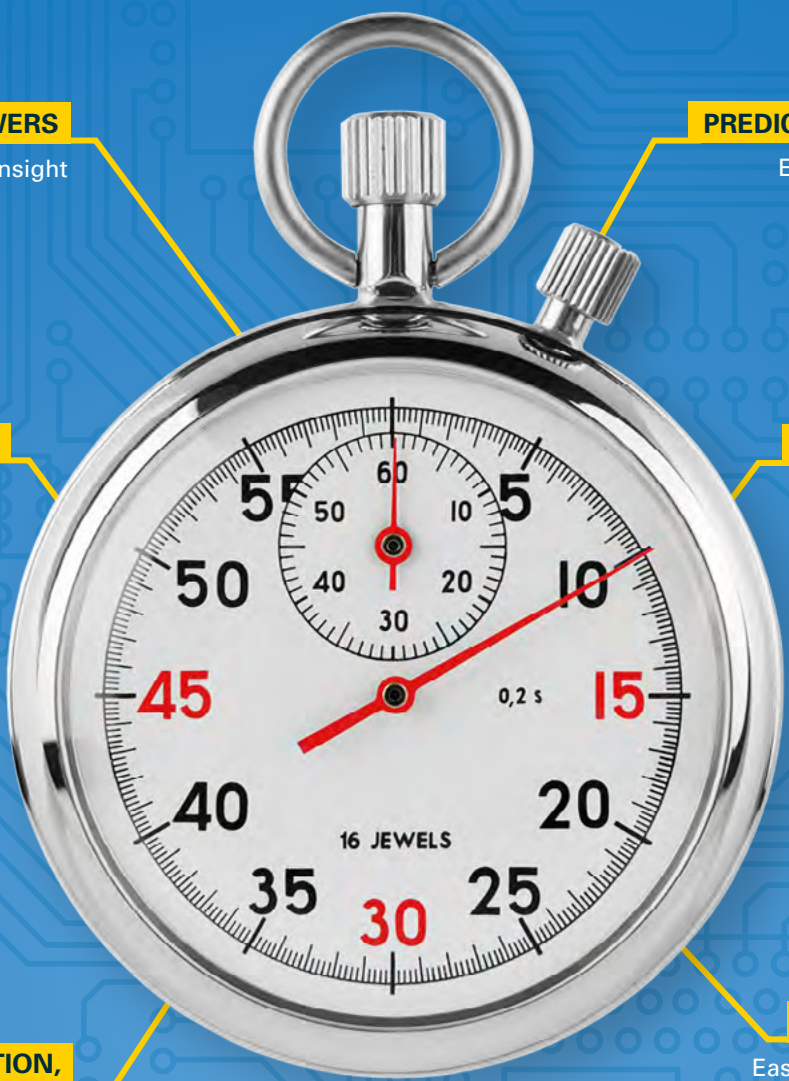
WHEN SECONDS MATTER... WE PROVIDE IMMEDIATE BUSINESS INSIGHT

LOW-LATENCY ANSWERS
Sub-second from source to insight

PREDICTABLE FAST QUERIES
Even as data volumes grow

HISTORICAL, FRESH DATA
Combined for deeper insight

POWERFUL ANALYTICS
Qualitative and quantitive



SCALABILITY, DISTRIBUTION, AND AVAILABILITY
Round the world and into the peta-scale

RICH VISUALIZATION
Easy to assemble dashboards and APIs

Acunu Analytics on Cassandra delivers instant answers from event streams, making it simple to build analytic applications and live dashboards that deliver fresh insight from live data.

Our unique approach of pre-building the cubes, on ingest, that satisfy the queries, means predictable, very short query times that are not dependent on data volumes.

There is no lag introduced by batching up events on arrival nor by indexing loaded data before it is available to queries. ***Results with millisecond latency.***

Read about our customers' success and hear them talk in our webinars at www.acunu.com

www.acunu.com | @acunu
contact@acunu.com | US office: +1 866-487-2650 | UK office: +44 203-1760143

Acunu

Everything You Need To Get Started with GODPAPER

The excitement of chess board game with the simplicity of flash application!
Godpaper is a chess board game framework for Flex. It enables developers to build flash chess games quick and faster, and it offers many helpful code libraries and helpers which speed up tedious tasks in Flex.

Godpaper is based on a modular design; meaning that you can implement specific libraries (i.e., game AI, data structure, plug-ins) at your discretion – which adds to the speed of the framework. This tutorial will attempt to introduce you to the basics of setting up the framework, including how to build a basic chess game that uses the MVC approach.

Why a Framework?
Frameworks allow for structure in developing applications by providing reusable classes and functions, which can reduce development time significantly. Some downsides to frameworks are that they provide unwanted classes, adding code bloat, which makes the app harder to navigate. From my perspective, a framework does several things:

- It makes it easier to work with complex technologies.
- It ties together a bunch of discrete objects/components into something more useful.
- It forces the team or individual to implement code in a way that promotes consistent coding, fewer bugs, and more flexible applications everyone can easily test and debug the code, even code that they did not write.

See more: *http://www.godpaper.com/godpaper/index.php/Framework*.

Why Godpaper?
Godpaper is a very powerful, well performing framework. While, it is perfect for a beginner (because of the short learning curve), it is also perfect for large and demanding Flash board games. Godpaper is developed by myself (Knight.zhou) and has thorough, easy to understand documentation. Below is a list of reasons of what makes Godpaper a smart framework to use?

- A modular collection of AS3 classes based on FLEX4 to simplify common tasks.
- A starting point for your FLEX applications.
- A demonstration of FLEX 4 best practices.
- A sophisticated component of FLEX collaboration project.

Why MVC?
For starters, MVC stands for Model, View, Controller. It is a programming pattern used in developing Flash board games (Figure 1). This pattern isolates the user interface and back-end (i.e. database) interaction from each other. A successful implementation of this let's developers modify their user interface or back-end without affecting the other. MVC also increases the flexibly of an app by being able to re-use models or views repeatedly). Below is a description of MVC.

- Model: The Model deals with the raw data and database interaction and will contain functions like

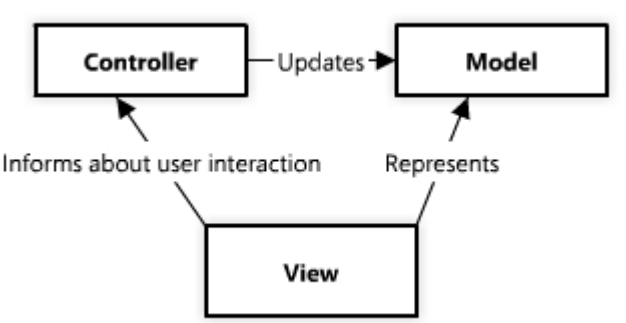


Figure 1. *Scheme*

adding records to a database or selecting specific database records. In Godpaper, the Model component is a collection of singleton class and value object, and can be included in the controller.
- View: The View deals with displaying the chess piece data and chess board and chess pieces interface controls to the user. In Godpaper, the view could be a chess board, pieces and gasket, view configure data or any other "virtual elements".
- Controller: The Controller acts as the in-between of View and Model and, as the name suggest, it controls what is sent to the View from the Model. In Godpaper, the controller is also the place to manage the game and chess pieces' statement.

An example of a MVC approach would be for a chess piece's movement in turn-based board game.

It is the human's turn now

- The user interacts with the view by dragging and dropping a chess piece.
- The controller (chess pieces manager) receives the chess-move data then applies it, meanwhile this data has been sent to the singleton class model and updated in the memory. Then send the turn flag to computer.

This turn is updated in the view and displayed to the user.

It is now the computer's turn now

- The game AI thinking and selecting chess piece to make a move.
- The controller (chess pieces manager) receives the chess moved data then applies it, meanwhile this data has been sent to the singleton class model and updated in the memory. Then send the turn flag to the human.

This turn is updated in the View and displayed to the user.

This may sound like a lot of work to do but, trust me when you are working with a large application, being able to reuse models or views save a great deal of time.

Let's start off!

Step 1: Downloading Godpaper

AS3,Stage3d (Starling): *https://github.com/yangboz/godpaper/tree/master/TheKnightErrant*.

FLEX4,MXML: *https://github.com/yangboz/godpaper/tree/master/TheRealKnight*.

Step 2: Installing and Exploring Godpaper

Once you have downloaded "YourChessGameTemplateProject.zip", all you need to do is unzip it, and rename the project and package folder to your application name or code name.

Now that its on your local file system, next I will explain what all the folders and files are for: Figure 2.

- The `src` folder stores all the files that make Godpaper work.
- The `assets` folder stores all the assets files relevant to the application UI interface. Which includes the background image of chess board, the skin class of chess piece, thumbnails etc.
- The `controllers` folder(com.godpaper.yourpackage) stores all the controllers for the application.
- The `factory` folder stores all the factory class for producing chess pieces and gaskets.
- The `managers` folder stores all the game and chess pieces' statement that are specific to your application.
- The `vo` folder is for chess piece's value object that maintains the functioning of chess movement.
- The `components` folder stores all the view components (board, piece, gasket) for the application.
- The `libraries` folder stores all the libraries (framework,plug-ins) that are specific to the application.
- The `Main.mxml` file is the part that does all the Godpaper magic. It also let's you configure the setting of the system and plug-in implementation.

Step 3: Testing Godpaper

We will do a quick test to see if Godpaper is up and running properly. Build and run, you should see the following (Figure 3).

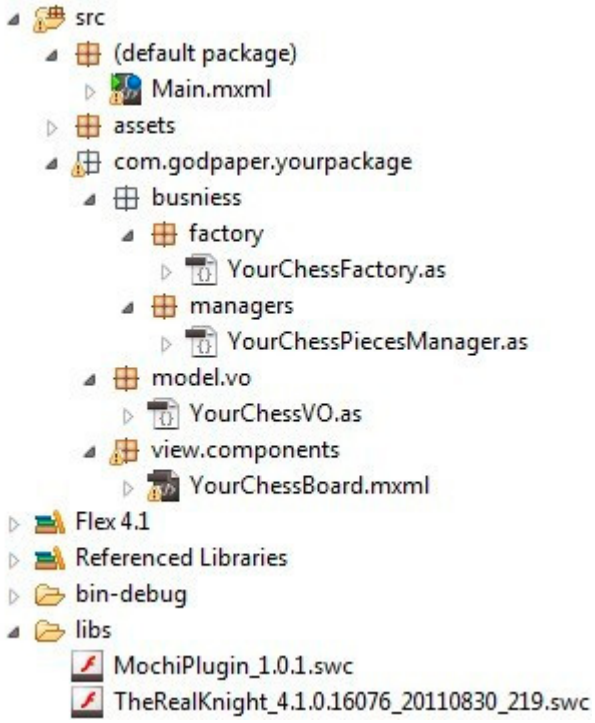


Figure 2. *Folders and files*

Step 4: Configuring Godpaper

If you are up and running, we should finish the configuration. We are starting to configure it specifically for our new Flash chess game. If you want to use a 3-by-3 grid with your application, (which in this tutorial we do.) open up src/Main.mxml and set the following variable items to its corresponding values. This code connects to a static class called "BoardConfig" on Godpaper framework `swc` package.

```
1. BoardConfig.xLines = 3; //the number of repeated lines
    in x axis.
2. BoardConfig.yLines = 3;//the number of repeated lines
    in y axis.
3. BoardConfig.xOffset = 100;//the offset value of each
    grid in x axis;
4. BoardConfig.yOffset = 100;//the offset value of each
    grid in y axis;
5. BoardConfig.width = 200;//always equal to (xLines-
    1)*xOffset.
6. BoardConfig.height = 200;//always equal to (yLines-
    1)*yOffset.
7. BoardConfig.xScale = 1;//the scale rate of board
    horizontally.
8. BoardConfig.yScale = 1;//the scale rate of board
    vertically.
9. BoardConfig.xAdjust = 50;//the adjust value for gasket
    start point in x axis;
10. BoardConfig.yAdjust = 0;//the adjust value for gasket
    start point in y axis;
```

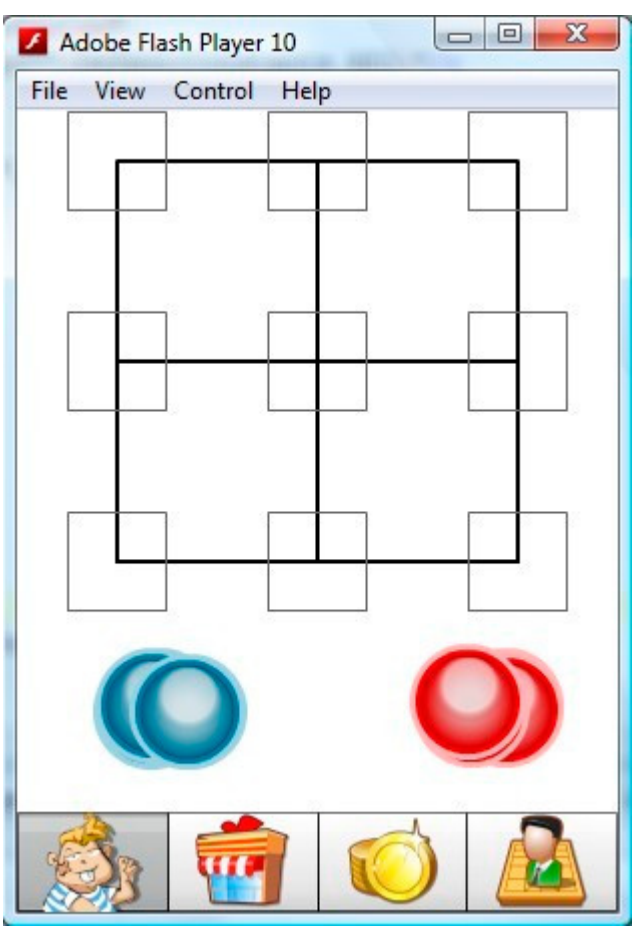


Figure 3. *Test*

Additionally, you can pre-define some connection conditions (such as TicTacToc,Connect4...) which included the connect direction and the number of connection;

```
1. BoardConfig.hConnex = true;//enable the horizontal
    connection.
2. BoardConfig.vConnex = true;//enable the vertical
    connection.
3. BoardConfig.fdConnex = true;//enable the forward
    connection.
4. BoardConfig.bdConnex = true;//enable the backward
    connection.
5. BoardConfig.numConnex = 3;//the number of connection.
```

Currently, the Godpaper setup will have a default controller (factory) called "YourChessFactory"; you can find this in the `src/com/godpaper/yourpackage/business/factory` folder. For this tutorial, rename them and corresponding to your package name.

```
1. PieceConfig.factory=YourChessFactory; //the default
    chess factory for your reference.
2. PieceConfig.scaleX = 1.0; //the scale rate
    horizontally.
3. PieceConfig.scaleY = 1.0; //the scale rate vertically.
4. PieceConfig.maxPoolSizeBlue = 5; //the total number of
    chess pieces(blue side).
5. PieceConfig.maxPoolSizeRed = 5; //the total number of
    gaskets(red side).
6. PieceConfig.usingDragProxy = true; //display the
    dragging proxy image.
```

Godpaper also has variable items to its corresponding chess gaskets configure values.

```
1. GasketConfig.borderVisible = true; //you can display
    this border for debugging.
2. GasketConfig.maxPoolSize = 9; //the total number of
    gaskets.
3. GasketConfig.tipsVisible = false; //you can display
    this position tips for debugging.
```

Do not worry about this series of variables, it is necessary and optional.

Step 5: Create the Chess Board View

The view file (Chess Board) is what the user sees and interacts with, it could be a segment of a page, or the whole page displayed on Flash player. To make the chess board view for our tutorial, there is a default file

named “YourChessBoard.mxml” in the “yourpackage/view/components” folder.

Next, we just need to create our normal chess board, only horizontal and vertical repeated lines, and then you will find that,”Degrafa graph framework” is easy to learn, easily extendable and very powerful.

In this stage, the “TicTacToe” chess board reference on “Grid Line Repeater”, once you have defined the “BoardConfig.xLines/yLines”, the default chess board view file (YourChessBoard.mxml) is bindable (Flex native bind features) with these variables, actually your chess board is already prepared.

Additionally, you can use this powerful tool to draw any geometry graph limited only by your imagination.

Step 6: Create the Chess Pieces Box View

At the first beginning of the type of chess board game, there are two main classes, full information and partial information. Let’s explain more detail on this.

- *Full information*, aka “Chess, Chinese Chess...”, to computer programming, at first the board information is full completely. (ApplicationDefault)
- *Partial information*, aka TicTacToe, Connect4,Go..., to computer programming, at first the board information is partial, as the game played as the board information `increased.(ApplicationDefaulter)`

In our Flex project template files, you can easily switch to the root name space (ApplicationDefault or ApplicationDefaulter) at “Main.mxml”.

For “TicTacToe”, the application’s name space is “ApplicationDefaulter”, in reality game, each player has their chess pieces box, where hold on each player’s pieces. In the same way, in our Flash chess game, we have two components to represent the box, also you can define a rectangle area where your pieces come up at first .

Remember we have defined the number of “PieceConfig.maxPoolSizeBlue/Red” as 5, which decided the game rules. At the first glance, we can see each pieces’ box has 5 pieces inside its “child Area”.

Now please put this two pieces box on the stage. We have the reality chess board and chess boxes with pieces!

Note: We have the same realistic way to move piece, just dragging one piece from box,then drop on the board.

Step 7: Create the YourChessVO Model

Models are important in Godpaper, and it is considered best practice to use model VO (value object). They are just AS3 classes that contain functions which work with game rules from the chess information. Go ahead and open the “YourChessVO.as” file.

In the `com/godpaper/yourpackage/model/vo` folder. In this file, was created a YourChessVO class, ChessVO-Base construct and a function called `initialization()` with row/column index and flag and identifier.

In the initialization function we are going to use “BitBoard” function which speeds up chess game rules development times when working with Godpaper and data structure.

Essentially, they are simplified functions to create bitboard. This idea inspired me comes from: http://chess.dubmun.com/bitboard.html.

To be a simple game bitboard (TicTacToe),this is our bitboard at initialization: (“.” means empty, “B” means blue chess pieces, “R” means red chess pieces);

.	.	.
.	.	.
.	.	.

We can make a representation of anything from this. For example:

Occupies

Spaces “occupied_red“ by red chess pieces: (TicTacToe game rule defined, at first, any empty cell can be filled);

R	.	.
.	.	.
.	.	.

Listing 1. *A longer piece of Component code*

```
1:PiecesBox id=”bluePiecesBox” //Using the default PiecesBox provider by Godpaper framework
left=”20” bottom=”50” width=”100” height=”100” //Define the constrained position.
backgroundImage=”” //Define the Pieces Box background image.
backgroundImageFillMode=”scale” //Define the Pieces Box background image style.
borderVisible=”false” //Trim the container’s border.
childrenArea=”{new Rectangle(20,0,25,25)}”//The birthplace of chess pieces.
type=”{DefaultConstants.BLUE}”/>//The type of chess box(Red or Blue).
```

Spaces “occupied_blue” by blue chess pieces: (TicTacToe game rule defined, at first,any empty cell can be filled);

.	.	.
.	.	.
.	.	B

So, “all_pieces“ = “occupied_blue“ or “occupied_red “;

R	.	.
.	.	.
.	.	B

Notes: There are no attack rules for “TicTacToe”;

Moves

Now “occupied_red“ logical “xor” with the “all_pieces“ bitboard and we get legal moves for these move_red.

X	.	.
.	.	.
.	.	X

Now “occupied_blue“ logical “xor“ with the “all_pieces“ bitboard and we get legal moves for these move_blue.

X	.	.
.	.	.
.	.	X

Captures

At this stage, the “TicTacToe” has no capture rules, but you should to apply this game rule to other games, such as “Chess”, easy logical and this bitboard with opposing piece positions to compute captures.

- “captures_blue“ = “ moves_blue“ and “occupied_red”;
- “captures_red “= “moves_red“ and “occupied_blue”;

Step 8: Create the YourChessFactory Controller

To apply all the rules received from the game rules, we put it in a “for-each” loop task that loops through all the virtual elements (chess pieces and gaskets). You may have noticed that we are using Cairngorm3 and Parsley task library files, this provides a convenient and time saving way to write echo statements. Let’s explain the detail tasks:

createChessPiece task

```
public function createChessPiece(position:Point,
                    flag:int=0):IChessPiece
```

createChessGasket task

```
public function createChessGasket(position:Point):IChes
                    sGasket
```

generateChessVO task

```
public function generateChessVO(conductVO:ConductVO):IC
                    hessVO
```

generateOmenVO task

```
public function generateOmenVO(conductVO:ConductVO):OmenVO
```

Step 9: Create the YourChessPiecesManager Controller

```
override protected function blueSideHandler():void
override protected function redSideHandler():void
```

Step 10: Distribution and monetization

And so on.

Summary

Briefly touched on Godpaper framework with a bare-bone project. Turn based game concept and bit-board data structure implementation;

On the Web

- *http://blog.lookbackon.com/* – Blog articles,
- *http://github.com/yangboz/* – Follow github.

Glossary

- If the article contains important and/or highly technical terms, I put them in this inset as a list.
- Chess Piece: Chessman;
- Chess Gasket: A place holder for chessman;Chess Board: At bridge table with chess gasket;

KNIGHT ZOU

The author (Knight.zhou), has a more than seven years working experience as a Software Developer, about three years R&D work at HP Ltd., as a quick leaner with passion for new edge technologies, intended primarily for developing AS3 Artificial Intelligence system.

Dr.WEB®
20 years

Dr.Web SplDer is 8-legged!

New Version 8.0

Security Space and Dr.Web Antivirus for Windows

Get your free 60-day license under https://www.drweb.com/press/
to protect your PC and your smartphone with Dr.Web!
Your promo code: ***Hakin9***

Protect your mobile device free of charge!
https://support.drweb.com/free_mobile/

Doctor Web is a Russian anti-virus vendor with a software development record dating back to 1992.
www.drweb.com

IOS Software Development with Adobe AIR

Adobe AIR is a great tool for porting games to mobile devices or even to design directly for them. AIR technology allows running your flash game on IOS, Android and yes ... BlackBerry with small modifications in your code.

Adobe AIR (Adobe Integrated Runtime) is a cross-platform run-time system developed by Adobe Systems. To develop AIR application you can use one of the following technologies:

- Flash / Flex / ActionScript
- HTML / JavaScript / CSS / Ajax
- PDF can be used as well with both

Apps developed with AIR can be launched on IOS, Android, or BlackBerry Tablet OS. But today we will only discuss the IOS segment. AIR apps can be published as native phone apps (for IOS it is .ipa). Now AIR 3.7 Beta is available for developers. Adobe did alot for this version and we have an ability to use the new SDK.

So, first download the latest SDK on this link *http://labs.adobe.com/technologies/flashruntimes/air/*.

Screen Resolution

It's great to create one graphic and display it on different screens. We can detect the device screen resolution with flash.system.Capabilities. This class provides properties that describe the system that is hosting an application. Now, we need to call just three methods to see the screen resolution and screen dpi. Save these values to constants of integer type (don't use type Number). Then we have the resolution in global constants. Using them, the resizing is much easier; also it's possible to get access to them in other places within the same class. The graphics can be added to a scene according to resolution or by resizing one set. See the Listing 1.

Please notice, that the screen resolution will be different in retina and non-retina mode. You can change this mode in the descriptor file which is named your_app_name.xml (this file is created automatically). To make

Listing 1. *Detect screen resolution*

```
private const xRes:int = Capabilities.screenResolutionX; //returns 640 pixels in width on Iphone 4 Retina
private const yRes:int= Capabilities.screenResolutionY; //returns 960 pixels in height on Iphone 4 Retina
private const dpi:int = Capabilities.screenDPI;//returns screen dpi
```

Listing 2. *Requested display resolution*

```
<iPhone>
     <requestedDisplayResolution>high</requestedDisplayResolution>
  </iPhone>
```

Listing 3. *Requested display resolution exclude devices*

```
<iPhone>
     <requestedDisplayResolution excludeDevices="iPad3  iPad4">high</requestedDisplayResolution>
  </iPhone>
```

your app work with retina mode, you should find tag `<requestedDisplayResolution>` and include value of that mode you are needing (high – for retina mode, standard – for non-retina mode; see Listing 2).

In AIR 3.6 was added new tag as child of element `<iPhone>`. This tag helps to switch different modes on different devices. In our example we have "high" resolution and have iPad3 and iPad4, so our app will be working on those devices in standard resolution mode (Listing 3).

Touch Input

We have an application and we need to be able to control it on devices which have no control buttons. For this issue, AIR allows you to listen and handle multitouch events. AIR can convert mouse events to touch events by default (for example mouse click, or dragging) which avoids rewriting code if there is no other necessery functionality. But AIR also has special multitouch events for handling any types of finger taps and gestures. So the first thing we need to do is to specify which types of finger taps will be handled. Multitouch.inputMode can take one of three properties:

- GESTURE – handles five events: `GestureEvent.GESTURE_TWO_FINGER_TAP`, `GesturePressAndTap.GESTURE_PRESS_AND_TAP`, `TransformGestureEvent.GESTURE_ROTATE`, `TransformGestureEvent.GESTURE_SWIPE`, `TransformGestureEvent.GESTURE_ZOOM`.
- `TOUCH_POINT` – handles eight events (of one type TouchEvent): `TOUCH_BEGIN`, `TOUCH_END`, `TOUCH_MOVE`, `TOUCH_OVER`, `TOUCH_OUT`, `TOUCH_ROLL_OVER`, `TOUCH_ROLL_OUT`, `TOUCH_TAP`.
- NONE – AIR just handle mouse events.

There are also methods for enabling touch dragging (Listing 4):

```
stopTouchDrag();
startTouchDrag();
```

Listing 4. *Using touch events*

```
package sdj
{
    import flash.display.Sprite;
    import flash.events.TouchEvent;
    import flash.ui.Multitouch;
    import flash.ui.MultitouchInputMode;
    public class Main extends Sprite
    {
        private const xRes:int = Capabilities.screenResolutionX;
        private const yRes:int = Capabilities.screenResolutionY;
        private const dpi:int = Capabilities.screenDPI
        public function Main():void
        {
            Multitouch.inputMode = MultitouchInputMode.TOUCH_POINT;
            drawRect();
        }
        private function drawRect():void
        {
            var rect:Sprite = new Sprite();
            rect.graphics.beginFill(0x00000, 1);
            rect.graphics.drawRect((yRes -100)/2, (xRes -100)/2, 100, 100);
            rect.graphics.endFill();
            addChild(rect);
            rect.addEventListener(TouchEvent.TOUCH_BEGIN, drag);
            rect.addEventListener(TouchEvent.TOUCH_END, stop);
        }
        private function drag(e:TouchEvent):void
        {
            e.target.startTouchDrag(e.touchPointID);
        }
        private function stop(e:TouchEvent):void
        {
            e.target.stopTouchDrag(e.touchPointID);
        }
```

Listing 5. *Set render mode*

```
<initialWindow>
    <renderMode>cpu</renderMode>
  </initialWindow>
```

As you can see, both drag methods use touchPointID values as parameters. What is touchPointID? This property is responsible for correct event handling. Because the devices have multitouch, we have to know when react. This property is more critical for `TouchEvent.TOUCH_MOVE` and drag methods.

Render Mode

This is probably the most important issue, since performance is dependant on the correct render mode in different cases. And when should each mode be used? CPU is used when an application mainly uses vector graphics. It can cause performance decrising when bitmap graphics are used in this mode.

If the application does not contain a lot of vector graphics you can safely use this mode. But if the application contains a lot of graphics and animation, it is better to convert it to bitmap "on the fly" and use the GPU mode.

You can certainly prepare a raster graphics in advance for different screens, but this method is less streamlined and requires more work, and also the. apk size will significantly increase.

For object optimization, cacheAsBitmap property can be used. This property needs to be set as TRUE just for display objects that have no: transformation, alpha changing, or nested animation. Those objects can only move on the stage. So be careful when using this property otherwise your application will be very slow. For objects that are transformed or have changed transparency, you can use the following: `my_mc.cacheAsBitmapMatrix = my_mc.transform.cocatenatedMatrix; my_mc.cacheAsBitmap = true;`. But this approach will only work in the GPU mode. You need to set the render mode in the application descriptor file (Listing 5). AIR 3.7 allows to force CPU render mode on required devices. You just need to include them between tags `<forceCPURenderModeForDevices>` (Listing 6).

Orientation

Changing screen orientation according to device orientation is important for usability. Please, take a look at the Figure 1 and remember all these orientations.

To determine when the device is changed, orientation AIR has flash.events.StageOrientationEvent. There are two ways we can re-orient stage objects: first – allow AIR do it itself; second – change objects size and orientation with Action Script 3.0. Let see the first technique (we will work with the app descriptor file). There are few cases are shown Listing 7-11. So, instead of using the last case I recommend to change the stage object properties programaticlly. Take a look at the code which handles all device orientations (Listing 12).

One app for Both iPhone and iPad

App can be deployed for iPhone (iPod Touch), iPad or for all those devices. Open the app descriptor file,

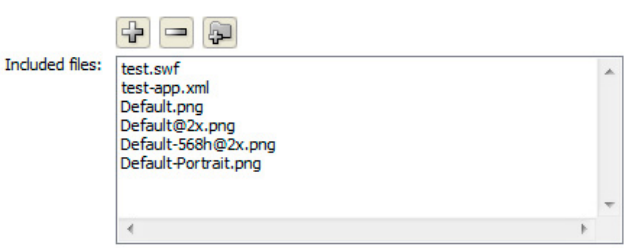


Figure 2. *Adding default images*

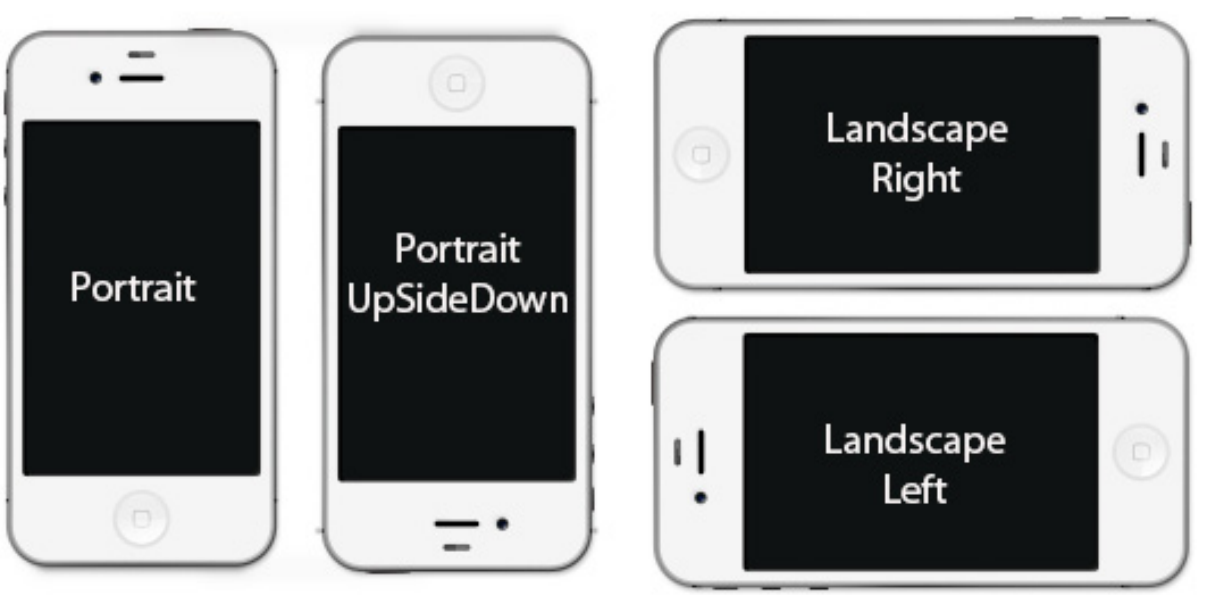


Figure 1. *Device orientations*

Listing 6. *Force CPU mode*

```
<renderMode>GPU</renderMode>
...
<iPhone>
   <forceCPURenderModeForDevices>iPad1,1 </
                    forceCPURenderModeForDevices>
</iPhone>
```

Listing 7. *Constantly oriented to LandscapeLeft*

```
<initialWindow>
    <autoOrients>false</autoOrients>
    <aspectRatio>landscape</aspectRatio>
  </initialWindow>
```

Listing 8. *Constantly oriented to Portrait*

```
<initialWindow>
    <autoOrients>false</autoOrients>
    <aspectRatio>portrait</aspectRatio>
  </initialWindow>
```

Listing 9. *Can be re-oriented to LandscapeLeft or LandscapeRight*

```
<initialWindow>
    <autoOrients>true</autoOrients>
    <aspectRatio>landscape</aspectRatio>
  </initialWindow>
```

Listing 10. *Can be re-oriented to Portrait or PortraitUpsideDown*

```
<initialWindow>
    <autoOrients>true</autoOrients>
    <aspectRatio>landscape</aspectRatio>
  </initialWindow>
```

Listing 11. *Can be re-oriented to all modes (not recommended)*

```
<initialWindow>
    <autoOrients>true</autoOrients>
    <aspectRatio>auto</aspectRatio>
  </initialWindow>
```

Listing 12. *Detecting device orientation changing*

```
package sdj
{
   import flash.events.StageOrientationEvent;
   public class Main extends Sprite
   {
      public function Main():void
      {
                stage.addEventListener(StageOr
                  ientationEvent.ORIENTATION_
                  CHANGE,OrientationChange);
      }

      private function OrientationChange(event:StageOr
                  ientationEvent):void
      {
         switch (stage.deviceOrientation)
         {
            case StageOrientation.DEFAULT:
            {
            // Default (Portrait) orientation.
            break;
            }
            case StageOrientation.ROTATED_RIGHT:
            {
            // LandscapeRight orientation.
            break;
            }
            case StageOrientation.ROTATED_LEFT:
            {
            //LandscapeLeft orientation.
            break;
            }
            case StageOrientation.UPSIDE_DOWN:
            {
            // PortraitUpsideDown orientation.
            break;
            }
         }
      }
   }
}
```

Listing 13. *Set device family property*

```
<iPhone>
    <InfoAdditions>
<![CDATA[<key>UIDeviceFamily</key>
<array>
<string>1</string>
<string>2</string>
</array>
<key>UIApplicationExitsOnSuspend</key><false/>]]>
    </InfoAdditions>
  </iPhone>
```

Listing 14. *Closing an application*

```
<key>UIApplicationExitsOnSuspend</key> <true/>
```

Listing 15. *Accelerometer example*

```
package sdj
{
   import flash.display.Sprite;
   import flash.events.AccelerometerEvent;
   import flash.sensors.Accelerometer;
   public class Main extends Sprite
   {
      private var accel:Accelerometer;
      public function Main():void
      {
         if (Accelerometer.isSupported)
         {
            accel= new Accelerometer();
            accel.addEventListener(AccelerometerEv
                  ent.UPDATE, accelUpdate);
         }
      }
      private function accelUpdate
                  (e:AccelerometerEvent):void
      {
         outputField.text = new String(e.accelerationX);
         outputField.text = new String(e.accelerationY);
         outputField.text = new String(e.accelerationZ);
      }
   }
}
```

find tag `<iPhone>` then `<InfoAdditions>`. There you can see CDATA tag. For supporting iPhone (iPod Touch) family there should be `<string>1</string>`, for iPad – `<string>2</string>`. We have both lines, so our app will work on all devices (Listing 13).

Exiting Application

When the Home button is pressed on an IOS device, the app frame rate drops to 0 (paused). If you need to completely close your app, set the `UIApplicationExitsOnSuspend` property to true (Listing 14).

Launch Image

While the app is loading the user will see a black screen. To avoid this, you can load an image that will displayed until your app is completely loaded. Figure 2 shows how to add launch images to .apk using Adobe Flash Froffesional CS 5.5. Open the AIR IOS Settings window and include images.

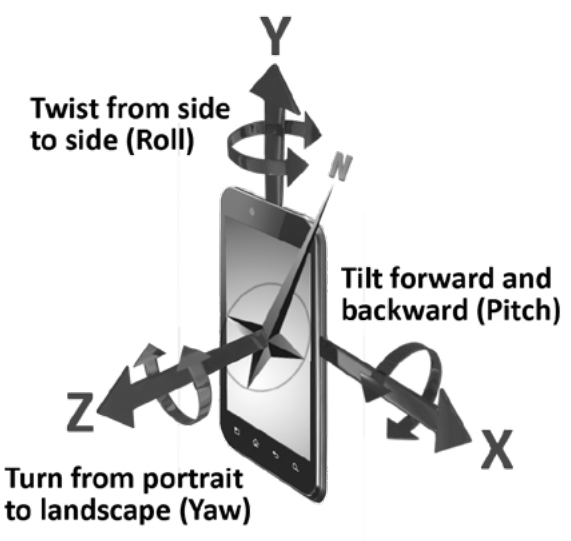


Figure 3. *Accelerometer demonstration*

Source links

- *http://help.adobe.com/en_US/air/build/WS901d38e593cd1bac1e63e3d129907d2886-8000.html#WS901d38e593cd1bac58d08f9112e26606ea8-8000*
- *http://spb-global.com/uncategorized/invensense-introduces-the-worlds-first-motionapps-platform-for-embedded-system-developers/*

Accelerometer

Accelerometer is widely used for game control. So AIR also has Accelerometer API. To work with it, we need two classes: flash.sensors.Accelerometer and flash.events.AccelerometerEvent. Accelerometer axis as shown Figure 3 and Listing 15.

Summary

So, we have examined the basics of AIR technology. These are not all the features this platform provides. For more detailed information, visit the Adobe website. I will not say this approach is better than using Objective C, because it would not be true. But this technology has advantages for some tasks and in certain situations. If you have your own game written in Flash, you can try to run it on IOS. It does not take much time and you can immediately see the result. Have fun!

SOSENYUK OLEG

The author is from Lutsk, Urkaine. Oleg works independently, as a Freelance Flash Developer.

Table 1. *Sizes of launch images*

File name	Image size	Usage
Default.png	320 x 480	iPhone, standard resolution
Default@2x.png	640 x 960	iPhone, high resolution
Default-568h@2x.png	640 x 1136	iPhone, high resolution
Default-Portrait.png	768 x 1004 (AIR 3.3 and earlier) 768 x 1024 (AIR 3.4 and higher)	iPad, portrait orientation
Default-Portrait@2x.png	1536 x 2008 (AIR 3.3 and earlier) 1536 x 2048 (AIR 3.4 and higher)	iPad, high resolution, portrait orientation
Default-PortraitUpsideDown.png	768 x 1004 (AIR 3.3 and earlier)768 x 1024 (AIR 3.4 and higher)	iPad, upside down portrait orientation
Default-PortraitUpsideDown@2x.png	1536 x 2008 (AIR 3.3 and earlier) 1536 x 2048 (AIR 3.4 and higher)	iPad, high resolution, upside down portrait orientation
Default-Landscape.png	1024 x 768	iPad, left landscape orientation
Default-LandscapeLeft@2x.png	2048 x 1536	iPad, high resolution, left landscape orientation
Default-LandscapeRight.png	1024 x 768	iPad, right landscape orientation
Default-LandscapeRight@2x.png	2048 x 1536	iPad, high resolution, right landscape orientation

What do all these have in common?

They all use Nipper Studio

to audit their firewalls, switches & routers

Nipper Studio is an award winning configuration auditing tool which analyses vulnerabilities and security weaknesses. You can use our point and click interface or automate using scripts. Reports show:

1) Severity of the Threat & Ease of Resolution

2) Configuration Change Tracking & Analysis

3) Potential Solutions including Command Line Fixes to resolve the Issue

Nipper Studio doesn't produce any network traffic, doesn't need to interact directly with devices and can be used in secure environments.

SME pricing from £650 scaling to enterprise level

evaluate for free at www.titania.com

2012 Computing Security Awards
WINNER
Enterprise Security Solution of the Year

2012 Computing Security Awards
WINNER
Network Security Solution of the Year

2012 Computing Security Awards
Runner-up
SME Security Solution of the Year

titania nipper STUDIO

CLOUD

www.titania.com
T: +44 (0) 1905 888785

RFC for a New Init Mechanism

(Another Way to Initialize Objects in Objective C)

Writing one's own initialization methods in an Objective-C program while respecting the Apple – originally NeXT – official writing rules can be a tedious task.

This article aims at offering another, simpler and more generic approach, inspired from scripting languages like Python or Ruby.

Let's start with a short reminder of the current and historical systems. For initialization methods to be added to one of our classes, three rules should be complied with:

- Redefining the designated initialization method of the superclass is mandatory only if the class defines a new one (with a different prototype).
- If a variant of an initialization method differs from the original only by the number of expected parameters (and not by their nature), then the method that accepts a 'low' number of parameters has to call one that accepts just a little more.
  (Implication: all initialization methods have to call directly or indirectly the designated initialization method of their own class.)
- The designated initialization method of a class must call the designated initialization method of its superclass.

For those familiar with initialization issues, remember how boring it can be to have to choose between two exclusive methods which can both apply to becoming the designated initialization method? Example:

- `(id) initProbeName: (NSString) *aName temperature InDegreesFahrenheit: (double) aTemperature`
- `(id) initProbeName: (NSString) *aName temperature InDegreesCelsius: (double) aTemperature`

Now imagine there is a way to write your initialization method only once, then use it in all your subclass hierarchies.

The principle is simple and understated: it consists in providing one unique method to perform all initializations, whatever the properties you want to define.

Furthermore, you don't have to create new initialization methods, again and again, every time you add a new class.

Implementation

First idea:

```
MyObject (subclass of NSObject)
```

Intuitively, the first idea that comes to the mind of an object-oriented developer who wants to add behaviors to a class is to 'subclass'.

In this particular case, we would have to subclass NSObject. That would probably work but that method is both inelegant and 'heavy':

- we are inserting a new class into an already complex hierarchy, and we must keep in mind from now on that we must prefer our own root class (rather than the official one, namely NSObject). Much ado about a tiny method…
- ii order to remain consistent, we would have to edit some of our parent class declarations for projects we want to maintain and in which we want to add our new mechanism.

This issue is conveniently solved by the concept of Category. Categories play at least three roles, the most prominent of which – and that is the heart of the issue at hand – is to enable developers to expand the list of a class' methods (no new properties) without requiring that the developer own the source code.

So this is the choice we retained.

Better integration, NSObject+MyInit (category):

For "not so modern" Objective C: Listing 1 and listing 2. Only for "modern" Objective C: Listing 3 and Listing 4.

Few lines but a compendium of concepts (some recents and other more historical):

- Category (to augment a class whose, in this case, we do not have the source code) – simply with the syntax NewClassName (CategoryName),
- fast iteration – with for / in construction (I would prefer @for / in to show it's an Objective C exclusiveness),
- KVC (Key-Value Coding) – by using here the "setValue:forKey:" method,
- "block" – availability of blocks is not really "modern" but it is always interesting to show usage of this feature,
- exceptions handling – with @try / @catch construction and a NSException instance, => a lot of controversy about NSException use on iOS (discussion in *http://stackoverflow.com/questions/4310560/usage-of-nsexception-in-iphone-apps* with a link on a full article: *http://club15cc.com/code/objective-c/dispelling-nsexception-myths-in-ios-can-we-use-try-catch-finally*)
- modern dictionary manipulation syntaxes (incidentally of NSArray lists): initialization (container literals), access (subscripting),
- pseudo type instancetype (which is an efficient alternative of id for this kind of methods),
- more below…

Listing 1. *file NSObject+InitWithDictionary.h*

```
@interface NSObject (InitWithDictionary)

- (id) initWithDictionary: (NSDictionary *) aDict;

@end
```

Listing 2. *file NSObject+InitWithDictionary.m*

```
#import "NSObject+InitWithDictionary.h"

@implementation NSObject (InitWithDictionary)

- (id) initWithDictionary: (NSDictionary *) aDict
{
    if (self = [self init])
    {
        [aDict enumerateKeysAndObjectsUsingBlock: ^(id
                    iVarName, id initValue, BOOL *stop)
        {
            @try
            {
                [self setValue: initValue forKey:
                    iVarName];
            }
            @catch (NSException *exception)
            {
                NSLog(@"initWithDictionary assign
                    error (%@) value %@", [exception
                    reason], initValue);
            }
        } ];
    }
    return self;
}

@end
```

Listing 3. *file NSObject+InitWithDictionary.h*

```
@interface NSObject (InitWithDictionary)

- (instancetype) initWithDictionary: (NSDictionary *) aDict;

@end
```

Listing 4. *file NSObject+InitWithDictionary.m*

```
#import "NSObject+InitWithDictionary.h"

@implementation NSObject (InitWithDictionary)

- (instancetype) initWithDictionary: (NSDictionary *) aDict
{
    if (self = [self init])
    {
        for (NSString *iVarName in aDict)
        {
            @try
            {
                [self setValue: aDict[iVarName]
                    forKey: iVarName];
            }
            @catch (NSException *exception)
            {
                NSLog(@"initWithDictionary assign
                    error (%@) value %@", [exception
                    reason], aDict[iVarName]);
            }
        }
    }
    return self;
}

@end
```

I detect the begining of a controversy. With such a mechanism the user (the consumer) will reap the error messages (resulting from the exceptions).

That is why it is important to use wisely the NSString constants (which carry our property names). This will never be a panacea, but prevent a significant proportion of inevitable typographical errors.

Question: how-to assign default values to properties during object instanciation?

Probably 2 ways. I don't know at that time which one is the best (if there is one). Start redefining initWithDictionary: which begins to send message [super initWithDictionary: aDict] then continue giving initial values to properties.

The other way you have to assign initial values for your objects is to redefine your own init method which – of course – starts with the message [super init]. It's possible simply because the main method, initWithDictionary:, send message to self.

In fact I thinks both solutions are not really equivalent and meet two different needs. (Up to you to discover which one.)

And if it really is unavoidable, I do not see any contra-indication to mix the two principles (historical initialization

Listing 5. *file Person.h*

```
#import "NSObject+InitWithDictionary.h"

@interface Person : NSObject

@property (copy, nonatomic) NSString *lastname;
FOUNDATION_EXPORT NSString * const PROPNAME_LASTNAME;
@property (copy, nonatomic) NSString *firstname;
FOUNDATION_EXPORT NSString * const PROPNAME_FIRSTNAME;
@property (assign, nonatomic) char gender; //
                    'F'emale, 'M'ale, 'U'nknown
FOUNDATION_EXPORT NSString * const PROPNAME_GENDER;
@property (assign, nonatomic) int age;
FOUNDATION_EXPORT NSString * const PROPNAME_AGE;

- (id) init;
- (void) setAge: (int) newAge;

@end
```

Listing 6. *file Person.m*

```
#import "Person.h"

@implementation Person

@synthesize lastname = _lastname, firstname = _firstname,
                    gender = _gender, age = _age;

NSString * const PROPNAME_LASTNAME = @"lastname";
NSString * const PROPNAME_FIRSTNAME = @"firstname";
NSString * const PROPNAME_GENDER = @"gender";
NSString * const PROPNAME_AGE = @"age";

- (id) init
{
    if (self = [super init])
    {
        self.gender = 'U';
        self.age = -1;   // -1 for unknown age
    }
    return self;
}

- (void) setAge: (int) newAge
{
    if ((newAge == -1) || ((newAge >= 0) && (newAge <
                    150)))
    {
        _age = newAge;
    }
    else
    {
        NSLog(@"Age off limit (%i)", newAge);
    }
}

@end
```

Listing 7. *file main.m*

```
#import "Person.h"

int main (int argc, const char * argv[])
{
    @autoreleasepool {
        Person *yannick = [[Person alloc]
                    initWithDictionary: [NSDictionary
                    dictionaryWithObjectsAndKeys:
                    @"Yannick", PROPNAME_FIRSTNAME,
                    @"Cadin", PROPNAME_LASTNAME,
                    [NSNumber numberWithInt: 45],
                    PROPNAME_AGE, nil]];

        NSLog(@"Welcome to %@ %@ who is %d years
                    old", [yannick firstname], [yannick
                    lastname], [yannick age]);
    }
    return 0;
}
```

method prototype and call to the method described here). Such a situation could arise in the context of a Protocol.

If it requires the definition of a method, for example `initProbeName: tempatureInDegreesCelsius`, then why not write:

```
- (instancetype) initProbeName: (NSString)
*aName temperatureInDegreesCelsius: (double) aTemperature
{
... optional checking (like if (aTemperature > 1000) ...)
return [self initWithDictionary: @{ @"name" : aName,
                    @"temperature" : @(aTemperature) }];
}
```

The syntax @(count) is a good example of what the last release of Objective C admits, it's called boxing.

Usage

For "classic" Objective C: Listing 5-7. The idea offered here became even more interesting with the recent changes in Objective-C. In particular literals and container literals which permit a very short wrtiing of many expressions, especially NSDictionary instances. The same for NSArray, NSNumber, boolean constants, etc. *http://clang.llvm.org/docs/ObjectiveCLiterals.html*. For "modern" Objective C: Listing 8-10.

Listing 8. *file Person.h*

```
The same as for "classic" Objective C, only replace
(id) init;
by
(instancetype) init;
```

Listing 9. *file Person.m*

```
The same as for "classic" Objective C, only replace
(id) init
by
(instancetype) init
AND… You can remove all the "@synthesize" lines.
```

Listing 10. *file main.m*

```
#import "Person.h"

int main (int argc, const char * argv[])
{
    @autoreleasepool {
        Person *yannick = [[Person alloc]
initWithDictionary: @{ PROPNAME_FIRSTNAME :
@"Yannick", PROPNAME_LASTNAME : @"Cadin", PROPNAME_
                    AGE : @45 } ];

        NSLog(@"Welcome to %@ %@ who is %d years
                    old", yannick.firstname, yannick.
                    lastname, yannick.age);
    }
    return 0;
}
```

End of concept overviews:

- dot notation (not so recent).
- automatic (implicit) synthesize.
- boxing (in an example above).
- modern syntaxes (literals) of constants writing (NSNumber, booleans, etc).

Depending the release of Objective C you want to write for, you have to use the first syntax or you can prefer the second (for the latest releases). My own opinion (thanks to add yours ;-)

Pros

- simplicity.
- the KVC mechanism allow keeping custom initialization of each variable. It's not a short circuit.

Cons

(performance) slowness (but this needs to be qualified because it probably depends on ratio: class hierarchy depth – given the potentially large number of messages – from the number of values to allocate / initialize) the main reasons for the poor performances: NSExceptions management and use of the KVC mechanism.

For interested readers: *https://github.com/nicklockwood/BaseModel*. For discussion about const in Objective C: *http://stackoverflow.com/questions/538996/constants-in-objective-c*.

YANNICK CADIN

Yannick Cadin is 45 years old, including 27 devoted mainly to computer industry (hardware and software). Programming of UBI-SOFT's very first game as freelance developer. Many salary jobs in small or medium companies: video game publisher, professional software editor, textile CAM software editor, Decision Tools software editor, computer resellers or distributors, consulting companies, value-added resellers (VAR), mapping company and even in the public sector with the title of Chief Operating Officer at the Louvre Museum. Manager of MICRO REPONSE specialized in providing computers running NEXTSTEP / OpenStep. Positions held: most of them, sales engineer, developer, support technician, trainer, ... Freelance writer on a more or less regular basis for a dozen magazines for the last 26 years. Casual proofreader and speaker. Red Hat Linux, Ubuntu, LPI, * BSD and Apple certified. Currently Manager of Diablotin.

SproutCore on the iPhone

Cocoa-in-JavaScript Comes of Age

Apple's other iOS app vector, the Open Web, is finally catching up.

There's a reason that Cocoa, Apple's flagship set of application development frameworks, is written, essentially, in C.

Cocoa, like any world-class framework, is a beast of abstraction: it exists to make the complex simple. Take Key Value Observing (KVO). This mainstay of MVC and related patterns allows you to specify that two properties, on two different, disparate objects, should have the same value. If one of them changes, the other one updates. And you can specify that certain actions should be triggered whenever the value changes. This lets you do trivial things like change a background color when the user chooses different options, or crucial things like recalculate downstream values when a user updates an upstream one.

It's pretty hard to understate the impact that KVO can have on your code. When values automatically stay synced, then different parts of your application don't need to know about each other. Separation of concerns, code sharing, all kinds of Best Practices become Easy Practices. And, once you've set up your binding, triggering that cascade of updates is as simple as this:

```
[obj setValue:newValue forKey:@"propertyName"];
```

It's like magic!

Of course it's not actually magic, it's abstraction at work, and it comes with a cost. That one easy line of code wraps dozens or hundreds of internal commands and complex data structures. All of the bound properties that need syncing, all of the observing methods that need calling, they all need to be tracked, updated, and properly disposed of when the time comes. So suddenly one simple command – a basic KVO setter – is triggering an invisible avalanche. Good abstractions come with the cost of allowing developers to *run* tons of code without the bother of having to *write* tons of code.

When you've got layers of abstraction serving as code multipliers like that, the underlying code needs to be *fast.* And it needs to be even faster to run on the iPhone, with its memory restrictions and puny CPU. So, with its eyes firmly on speed, Apple has bullheadedly refused to follow industry trends towards heavily interpreted languages, and even its experiment with garbage collection was canceled just a generation after it began. Cocoa remains in Objective-C, which is a speedy superset of C, and as uninterpreted a language as you'll find. And it needs to be, in order to stay fast when there's so much abstraction in the air.

Pan over to JavaScript. It's loosely-typed. It's garbage-collected. It's not even shipped around in semi-efficient intermediate representations: it's literally fed into the browser as raw text. For all of its vaunted flexibility, and for all of the billions of dollars that Google et al. have sunk into speeding it up, all of that still comes with a healthy performance penalty.

SproutCore, the granddaddy of JavaScript MVC frameworks, is Cocoa for the web. KVO and bindings are baked in at the lowest levels, and the SproutCore equivalent of our Cocoa setter from above – `obj.set('propertyName', 'newValue');` – triggers its own invisible, sophisticated avalanche. (Anyone who's peeked under SproutCore's hood in the Chrome debugger has seen this avalanche in action.)

That abstraction is extraordinarily powerful: SproutCore uses it to bring sophisticated application development strategies and sensibilities to the web application arena, a space which is often sorely lacking in those sensibilities. A SproutCore application will have well-organized, well-segregated models, views and controllers, managed by a powerful state machine and tied together with the magic of KVO. In short, a SproutCore

application will be an *application.* It just so happens to run in the browser.

The reasons to choose the web for your applications, if you can, are myriad, and well-established. There's the true cross-platform support that you get almost for free. There's the zero-hassle deployment, and an upgrade procedure for your users that's just one step long. ("Click refresh.")

But note the caveat – "if you can", and performance was a longstanding barrier. Eight years ago, JavaScript was too slow a language to allow for the valuable abstractions that you need to build a Cocoa-inspired framework. But Google marched its famous march towards faster JavaScript engines, and we've entered a golden era on the desktop.

The mobile space took a little bit longer. When the iPhone debuted, its JavaScript engine was pathetic. Keep in mind that the web was Steve Jobs' first choice for third-party apps, and the company's answer to anyone upset by App Store policies. For years, it was a laughable half-kept promise: Mobile Safari wasn't fast enough to run the New York Times smoothly, much less handle deep layers of valuable, application-y abstractions.

Then came the iPhone 4S and we were almost there, and then came the iPhone 5, and finally Steve Jobs' promise that we would have a non-curated, non-closed, cross-platform avenue to publish our apps on the iPhone has come true. Applications, real ones with grownup application architectures, ones that just happen to be written in JavaScript and happen to run in a web browser, can finally run on your phone, and it's only going to get better from here. You can check out one great mobile app at read.kobo.com, and lots of others are coming.

But the speed of a line of code, or the speed of a mathematical operation, isn't now and wasn't ever the point. Real applications need real application frameworks like SproutCore, and real application frameworks need room to abstract. They need enough headspace to hide massive complexity behind delightfully simple commands. It's 2013, and the web on the iPhone can finally deliver. Time to start creating.

DAVE PORTER

Dave Porter is a lover of JavaScript and a core team member of the SproutCore open source project. He hails from Boston, where he does contract development work through Appnovation, a Vancouver-based open source powerhouse. You can find him on Twitter @davecporter, and on GitHub at dcporter.

Integration Patterns

for Native Mobile App Development

Many native mobile apps live or die based on the APIs that serve them. Whether consuming third-party APIs or those delivered within your own enterprise, what's the best way to get A talking to B?

In this article I'll mainly be comparing, and contrasting, two architectural patterns for integrating Native mobile applications with back-end services; the "Mashup" App and the Native Adapter API.

'Mashup' Apps

A 'Mashup' App is one of the most common ways of integrating Native Apps with service providers. In this pattern the app integrates independently, and directly, with various backend services and APIs (these may be large scale third-party APIs, bespoke APIs delivered by partners, or APIs delivered within your own organisation) (Figure 1).

In the last 12 months I've noticed a steady rise in the amount of clients where architects are assuming this pattern and therefore that Native Apps (delivered internally) will directly consume "Strategic APIs", also delivered within the enterprise.

In this pattern the App is responsible for meeting the contracts of each API, filtering data to meet the requirements of the application, and deciding how to present the data for the particular audience.

Native Adapter APIs

A Native Adapter API is a dedicated component that provides services tailored towards one or more individual native apps (in architectural terms this can actually provide any, or all, of the functions of an Adapter, Bridge, Facade or Proxy (*http://en.wikipedia.org/wiki/Design_Patterns#Patterns_by_Type*) (Figure 2).

In this pattern there is a decision to be made about which responsibilities are fulfilled by the Native App and which by the Adapter API.

The three areas most affected by this choice are Performance, Security and Flexibility.

Performance

The precursor to any discussion on performance is to always remember that "Premature optimization is the root of all evil" (*http://en.wikiquote.org/wiki/Donald_Knuth*).

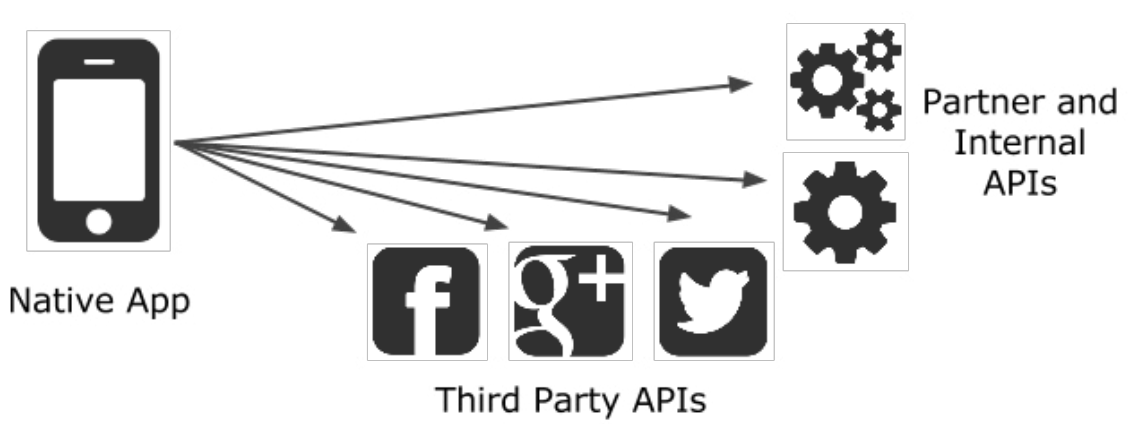


Figure 1. *Scheme*

With that out of the way let's talk about the performance benefits of Mashup Apps:

Scalability
If you are just consuming other peoples' APIs from a Native App then you are "scaling on the client-side" and have nothing to scale! The benefits of this should never be underestimated although many apps (particular being delivered by large enterprises) don't actually have this luxury.

Shortest Path
Introducing a Native Adapter API means that there is an "extra hop" when getting a request from the device, to the backend API and back. The great thing about the Mashup App pattern in this respect is that you know that it's not you introducing the bottleneck. That said, there are potential performance benefits to be had by introducing a component that is dedicated to making life easier for the app. This is particularly true if some of the APIs that you have to integrate with do not perform as well as you might hope! So, still bearing in mind our feelings on "premature optimisation", what options do a Native Adapter API enable?

Caching
If the nature of your app means that multiple users are making the same queries (or combination of queries ending in the same result) within an appropriately short period of time then caching can provide latency improvement for the later requests.

Caching can also be used to deliver improved resiliency by serving "stale" data if a backend is unavailable. This can help the app to maintain an acceptable level of service in the event of a provider outage. This can be achieved in combination with a back-end API by respecting HTTP Cache headers set by the API.

Prefetching
Prefetching is a kind of caching where the Native Adapter API may be able to predict one or two services that the app may require next. In this case the Native Adapter can make the predicted calls in advance of the user actually invoking the features and store the result in cache. This can deliver performance benefits over and above that which could be delivered by tuning the back-end API, but at the cost of wasting resources if the resulting calls don't materialise.

Bandwidth Reduction and Request Pruning
Focusing on the amount of data that is sent "over the wire" can significantly improve performance. This is especially true in the case of mobile devices, where it can also reduce the user's data costs. There are three main ways to reduce the bandwidth cost of a single request:

- Remove Irrelevance – The simplest way to reduce bandwidth, strip out any data in the response that the native app is not interested in.
- Provide Deltas – Where an app makes multiple requests to the same service but only a small part of the data has changed then the Adapter API can (by agreement) only send the changed parts. The ultimate case of this is sending a 304 (Not Modified) response if there has been no change at all.
- Compression – in most cases it isn't worth implementing custom compression but for very verbose data interchanges an Adapter can implement GZIP Content-Encoding if it is not already supported by a backend API. Most Native App libraries will happily handle decompression on the client side.

Request pruning simply means combining multiple requests into one. This should only be done if the app would require the result of both requests to react to the user (otherwise the performance of the app will degrade to the lowest performing provider).

Security
The main security problem associated with the "Mashup" App pattern is that "There are no secrets on the cli-

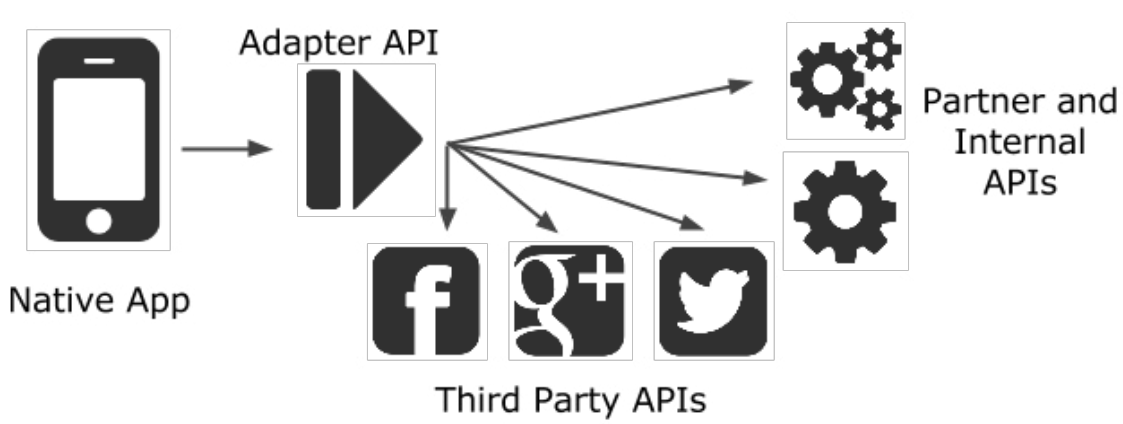


Figure 2. *Scheme*

ent-side". What this really means is that whatever you do to try and misdirect, hide or obfuscate what you are doing, everything that is done on a user's hardware, or sent to a user's hardware, can be observed by a user (whether that user is genuine, or an attacker!).

Keeping Secrets from an Attacker

A great example of needing to keep a secret is in an authentication scheme such as OAUTH2 (*http://oauth.net/2/*). If you are integrating with an API using this sort of scheme then you will be issued with a Client ID and a Client Secret. Typically, you redirect the user to a login/permission dialog (either on the web or in a native app/account manager) and then you are issued an auth code. You then use your Client Id and Client Secret, and the auth code issued to you, to get access to the APIs. The problem is with a pure client-side implementation is that you have no way of keeping your client secret a secret!. This means that no API provider (in their right mind!) will give you access to any privileged APIs unless you have a way of avoiding this issue. With a Native Adapter API, the client secret is kept securely on the server-side and so the problem is avoided.

Key Exposure – Denial of Service

Even in the case of an API that doesn't return user information, but is protected by an API key, a relatively trivial Denial-of-service (DoS) attack is possible when the API key resides on the client's device. If an attacker can obtain a copy of your API key from the device (via packet-capture, a 'jailbroken' device or reverse engineering of client-side code), then the attacker can automatically play a high-volume of requests directly against the API using your key to activate any rate-limiting or throttling, thereby denying service to genuine users of the app.

Content Filtering

In the case of partner and internal APIs, the information returned from the APIs can often be rather richer than an application needs. In the case of a web application, then this is simply filtered out on the server-side and so the user is never aware (this can be thought of as "Keeping secrets from the user"!). However, in client-side apps we know that there are no such things as secrets but an Adapter API can be used to "filter" the response from an API to only send information to the client that is actually required in the actual user view.

Whilst there are certain aspects of security that can only be achieved server-side, keep in mind that introducing any new component to a solution also introduces another potential vector of attack; this also applies to a Native Adapter API.

Flexibility

One of the main decisions for businesses, when deciding to support users accessing content or services via a mobile device, is Native Application vs Mobile Web (eg, HTML5/Responsive Design).

Often the two points that swing the decision in the direction of a Native app are (1) Discoverability (being able to be found in an app store has an enormous amount of value to a business) and (2) Native Interaction (either through actually needing 'native-only' features or deciding, rightly or wrongly, that there is a 'crispness' that can't be delivered using non-native technologies).

Once this decision is made, many projects entirely abandon many of the benefits that a 'mobile web' project can give (e.g. fast changing content, instant delivery of incremental updates, seamless bug fixes). When a Native Mobile App is consuming APIs directly then all logic relating to the feature set of the App has to be built into the app. This means that the only way changes can be made is by forcing upgrades of the app. This is never a "free" process as incremental versions of the app often introduce new testing overhead and process overhead (dealing with the Wheels of Cupertino!) as well as the overhead of convincing users to install new updates of the app.

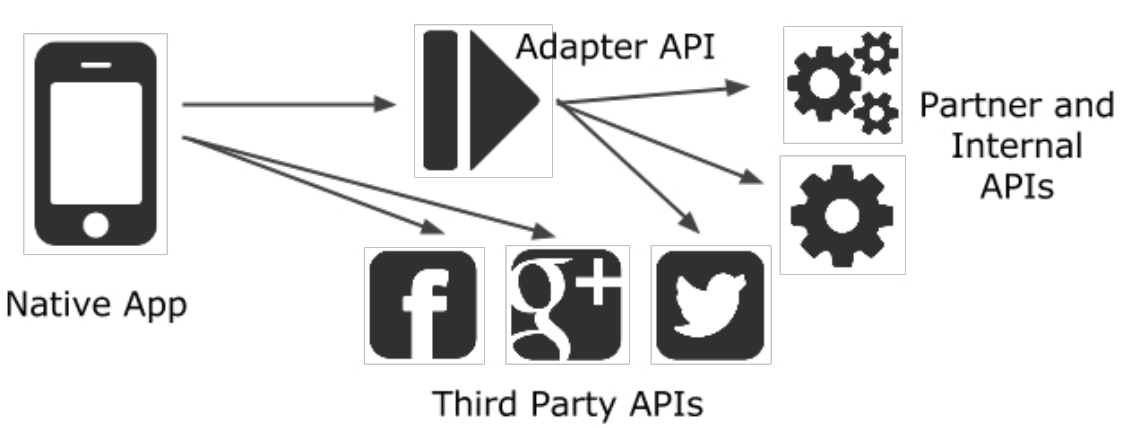


Figure 3. *Scheme*

A Native Adapter API can go some way to bridging the gap between a Native App and the benefits of Mobile Web:

Data-Driven Experiences

When using a Native Adapter API you have the option of pulling large amount of the presentation domain logic into the API. This could start off simply as string files for copy (allowing spelling corrections to be made without app releases) or error mappings and messages and continue to item ordering logic, and business rules that operate on the responses of multiple APIs.

Ideally logic data should be delivered to the app "lazily", so as not to negatively impact the overall user experience of the app. This could be done by combining logic data with actual data updates as well as possible background loading app logic updates and fixes after the app has delivered the main features to the users (just don't stick a spinner on the screen every time the app starts whilst downloading a whole new version of the app!)

"Shims"

Picture the scenario: You've just participated in the (highly publicised) release of a new app that consumes one of your organization's "Strategic APIs". Despite all of the automated testing that you had in place, you find out on launch day (after the app has gone to the app store and users have started downloading it) that certain high-res devices have a rendering issue causing any prices of 3 digits not to be visible (prices of 2 and 4 digits are fine!). What do you do?

At first, this seems to be a case of the Irresistible Force vs. the Immovable Object. The business needs this to be fixed as soon as possible, but updates to the app store will take time (and not necessarily reach all users). Well, we all know that the Irresistible Force vs. the Immovable Object is a paradox (*http://en.wikipedia.org/wiki/Irresistible_force_paradox*) and I can guarantee you that you'll be asked to "hack" your Strategic API so that it pads all 3 digit prices to 4 digits. As you probably already have other consumers of the API you don't want to change it for all consumers so you have to change the API to give a specialised (and incorrect!) response for a given consumer.... not looking so "Strategic" now is it!! This is just one scenario where a Native Adapter API can also "protect" inappropriate changes from "polluting" strategic APIs.

Shims can also be used to make short term fixes to data anomalies that come from back-end data, or to implement new business logic to meet short term deadlines that a Strategic API team might not be able to meet (but remember to give all of your automated test cases to the Strategic API team when they do come to implement the feature!).

In terms of flexibility it's hard to make any argument for a "Mashup" app offering more than a Native Adapter API.

In Conclusion

If you're an app developer with limited resources, then don't jump to start deploying Native Adapter APIs in the cloud, without considering what you are taking on.

However, if you are an enterprise shop, and are used to deploying and running server-side components, then don't forget to consider the benefits that a Native Adapter API could deliver.

Most importantly, this decision can be made on a per API basis leaving you with a hybrid model that should deliver the best of both worlds (Figure 3).

PHIL PARKER

Phil Parker is a software developer and architect (preferring Trenches to Ivory Towers!) for Equal Experts, one of the fastest growing technology companies in the UK. He specialises in lightweight, Agile, enterprise web application development, and has helped teams deliver server-side integrations for a number of "Top 5" iPhone and Android apps.

Mobile App Revenue

Generating revenues from iOS mobile application

The primary objective of any individual mobile App developer is to generate revenue. Building paid mobile apps (with a very low price tag) was the only way to generate revenues until 2009, unless it was built for some specific consumer/business, or offered as a service. This model only works when there is a significant number of downloads, preferably in thousands. It has been observed that the only a small fraction of the consumers are interested in buying premium (paid) apps.

The option of in App purchase was only available for the premium applications and, most of the consumers were not interested in those premium applications. In late 2009, Apple extended the in-App purchase option to the free applications also. It has changed the scenario drastically. The new advertisement platform, iAd was introduced in 2010. iAd allowed the developers of free apps to earn on their apps by displaying advertisements. The game has changed forever….

The expected audience of this article is the novice/inexperienced iOS developers.

Revenue Models

There are multiple ways to earn revenues from a mobile application:

Paid Applications

In this model, developers create a mobile application and sell it for a fixed amount to any potential consumer/business house or uploads it as a premium application in App Store (i.e. Angry Birds). The disadvantage of this model is that there are hundreds of thousands apps available in the App store and most of the consumers are not interested in purchasing premium application unless it is really offering something unique or really useful. Also significant marketing effort are required to convince the potential consumers to buy the app.

Freemium Apps

In the *Freemium* Apps model, developers create a basic version of an app, restricting many useful features. However an option is provided to the user to purchase those features from the application itself or through premium upgrades. This model is one of the most popular models followed by the mobile application developers. This is an age old concept, where companies/individuals offer to download a trial version of their new software or game, with the confidence that satisfied users will purchase the premium version at a later stage.

In App Advertisement

In the App Advertisement model, most of the Apps are free, where advertisements displayed through a "Banner" (shows a banner line on top or bottom of the application screen) or through Interstitials (an advertisement page appears during running the application). This is the most preferred model for free mobile App's because of the following reasons:

- Most of the Smartphone users download the free App instead of the paid Apps (the ratio is almost 400:1).
- In long run, the revenue generated from advertisements is more than the revenue generated from premium Apps, or from premium upgrades.
- In case of App piracy, the developers suffer from revenue loss. However, in case of Apps with advertisements, the issue of App piracy never occurs.

In this article, we shall limit our discussion only on In App Advertisement.

In App Advertisement Options

Once a developer decides to integrate Ads within their apps, the next step would be to choose the type of ads to be displayed. There are multiple Ad networks available in the market which allows the developer to inte-

grate third party advertisement into the App, without any trouble of collecting, shuffling the ads or counting the clicks and managing the invoices.

In the initial days of Apple iOS applications, the most popular advertisement network was offered by a company called AdMob. There is a SDK available for AdMob, which allows the user to integrate AdMob generated advertisement in the mobile applications. Later, AdMob was acquired by Google. Apple has come out with iAds, its own advertising solution. iAd framework is part of the iOS SDK now. At present, there are multiple mobile Ad networks:

- iAd – iAd is a mobile advertising platform developed by Apple for its iPhone, iPod, Touch, and iPad line of mobile devices, allowing third-party developers to directly embed advertisements into their applications. This platform is available only for Apple devices.
- AdMob – AdMob is one of the world's largest mobile advertising platforms. AdMob offers advertising solutions for many mobile platforms, including Android, iOS, webOS, Windows Phone and all standard mobile web browsers
- Few other leading Mobile Ad Networks are InMobi, Millennial Media, Brightroll, GreyStripe, Jumptap, MdotM, MobClix, ZestADZ, etc.

In this article, we shall limit our discussion to displaying advertisement through iAd. The overall concept remains same for each mobile advertising platform.

Similar to all other popular mobile advertising platforms, iAd facilitates integrating advertisements into applications sold from the iOS App Store. When the user taps on an iAd banner, a full-screen advertisement appears within the application, instead of opening the Safari Web browser. Once the users close the advertisement, they will return to exactly the position

Listing 1. *Sample code*

```
#pragma mark -
#pragma mark ADBannerViewDelegate methods

(void)bannerView:(ADBannerView *)
                    bannerdidFailToReceiveAdWithError:
                    (NSError *)error
{
  //hide thebanner
  [self moveBannerFromScreen];
}

(void)bannerViewbannerdidLoadAd: (AdBannerView *)
                    banner
{
  //show the banner when there is content
  [self moveBannerBackOn];
}
(void)moveBannerBackOn
{
  //bring back to screen
  CGRect theBannerFrame = self.bannerView.frame;
  theBannerFrame.origin.y = self.view.frame.size.
                    height - theBannerFrame.size.
                    height;

 //adjust existing table
 CGRect originalTableFrame = self.tableView.frame;
 CGFloat newTableHeight = self.view.frame.size.height
                    -  theBannerFrame.size.height;
 CGRect newTableFrame = originalTableFrame;
 newTableFrame.size.height = newTableHeight;
  self.tableView.frame = newTableFrame;
  self.bannerView.frame = theBannerFrame;

}

(void)viewDidLoadAd {
...
  //hide initially till sure there is an Ad
  [self moveBannerFromScreen];
...
}
```

Listing 2. *Sample code*

```
- (BOOL) bannerViewActionShouldBegin: (ADBannerView *)
                    banner willLeaveApplication: (BOOL)
                    willLeave
{
        //pause your App here
        ...
        return yes;
}

 //When the user comes back you get the callback:
- (BOOL) bannerViewActionDidFinish: (ADBannerView *)
                    banner
{
        //resume your App here
        ...
        return yes;
}
```

where they left their app. Initially Apple announced that apple will retain 40% of the Ad revenue. However, as of now, the developers get 70% of the Ad revenue.

In App Ads with iAD

IAd supports two different sizes of banner advertisement and a full screen advertisement format, called as interstitial. The banner ads are displayed at the top or bottom of the App area. Both portrait and landscape orientation is supported. iAds banner based advertisements are build using the ADBannerView class.

iAds interstitial advertisements are displayed in the full App area and normally used when users are moving from one screen to another within an application. Interstitial advertisements are created using the ADInterstitialView class. However, please note, this feature is only available for iPad.

Let's consider one simple example with the ADBannerView class. It is only a view and should be part of the view controller that you have in your app. This class manages, retrieves and displays the ads from the iAd network, managing the user interaction. The only thing we need to consider is:

- Placing the banner within view controller
- Responding to networking errors

So in Interface Builder, you need to drag the AdBannerView in the viewcontroller, in the same way we add other views onto our view controller. Be sure to add the iAd framework in the list of frameworks in XCode.

Next step will be to check the connectivity and other issues that might occur throughout during running the App. The banner queries the iAd network for a new ad, the internet connectivity is required. Also remember, the ADBannerView may not always have content in it, because of unavailability of network signals or the phone may be in the Airplane mode.

The two states that need to be checked through the ADBannerViewDelegate callbacks are:

- `bannerView:didLoadAd` has Ad content.
- `bannerView:didFailToReceiveAdWithError` in case of connectivity problems the ads are not received during this cycle.

If there is a 'fail', it is not critical. However, instead of displaying a blank banner area or empty box, one can hide it, and display again on the screen when there is content. A sample code snippet to show how this can be done: Listing 1.

Now the Ad will be displayed. We need to keep in mind that when the Ad is getting displayed on the screen, it is better to pause for some time to give the user the best experience when viewing the ad. Also the minimal states need to be saved. When the Ad is complete, the App can be resumed.

In `ADBannerViewDelegate`, when a user action begins on the Ad you get the following callback: Listing 2.

On the Web

- *https://developer.apple.com/iad/resources/*
- *https://developer.apple.com/programs/*
- *https://developer.apple.com/in-app-purchase/*

Summing it up

You can realize now, implementing iAd in your iOS application is very easy. Thousands of developers are using iAd in their free mobile Apps to generate long-term revenues. One big advantage of iAd is its integrated within the iOS. The developers does not need to bother about downloading and integrating another SDK. Debugging is easier. Also they do not need to worry about the support of the SDK they are using in the next version of iOS. There are many other useful features like, blocking competitors Ad or filtering unwanted Ad's. However, those are beyond the scope of this article.

TRIDIBESH DAS, PMP®, CSM®

Tridibesh Das has been associated with the IT Industry for last two decades. He started his IT Career as a mainframe programmer, moved to client-server technologies, then to the cloud, and now is involved in mobile technologies. He loves this agile and ever evolving IT Landscape. He is presently working in Interra Information Technology Inc., as Director, Engineering.

App Wings™

www.appwings.com

anand™ tech media

www.atmedia.in

iOS App Development Methodology – S2C2

This article describes an iOS App development methodology called S2C2. The intent of this methodology is to optimize iOS app client development by means of a standardized approach. You will require Xcode to follow along with the sample app creation.

S2C2 is a play on the abbreviation SSCC. This simple abbreviation covers the scope of the four-step app development methodology: Start It, Stub It, Connect It, and Complete It.

These four steps take you through the development of an iOS app. Here is a quick summary of the steps:

- Start It: Beginning the project, selecting templates, and creating project files.
- Stub It: Stubbing properties/methods in the .h/.m files in XCode to prepare for work in Interface Builder.
- Connect It: Creating the UI and connecting stubbed properties/methods with Interface Builder.
- Complete It: Completing the .m code for the stubbed methods in XCode.

Once these core steps are applied and repeated as desired, you simply build & run, and you are done!

S2C2 Methodology Step by Step Example

The objective of this step by step example is to create the quintessential Hello World app on an iOS device. This app has a single View Controller containing a Text Field and a Label. The Text Field will allow you to capture user input via the keyboard, and the Label will be used to display the text entry result. In order to achieve these objectives, we will build an application that does the following:

- Create and expose a View Controller/View to house our UI Controls
- Expose a Text Field that allows user to type something via a keyboard
- Display what the user has entered in a Label

The following controls will be used to create the app:

- Controllers
  - UIViewController
- Input/Display Controls
  - UITextField
  - UILabel

Start It

For iOS development, there are two primary facets to the IDE, the XCode IDE for coding and Interface Builder for UI Development.

An important concept in iOS development is connecting the objects between the Xcode class files (.h/.m) & Interface Builder files (.xib). Both sides (.h/.m & .xib) must be bridged for connectivity to one another. This ensures that the correct properties/methods are associated/referenced during code execution of the .m file. We will demonstrate these connections with a UITextField, and a UILabel.

Create the Project

To get started – open Xcode, create a new Project (File/New Project):

- Type: Window-based Application
- Name: SDKtoday101

Technically, at this point, you've just created your first iOS app! Really – it's that simple. Select Build & Run to test the app in the simulator for yourself.

Ok, so it runs, but admittedly, it doesn't really do anything – so lets continue with the example. In order to write code to interact with properties/methods in Xcode, we have to connect them from the UI to the code.

Before we can do that, we need to define the properties/methods. For this project, we will create a new View Controller class. To do this, select File/Add File, and choose the following options:

- Cocoa Touch Class Template
- Select UIViewController subclass

Set the name/options:

- Name
  - SDK101ViewController.m
- Options
  - UITableViewController subclass: No
  - With XIB for user interface: Yes
  - Also create .h: Yes

This concludes Step 1 – Start it.

Stub It

In this section we will work exclusively in the Xcode IDE. The objective is to define or stub the properties & methods in the code files (.h/.m). This work is done in preparation to connect them to the XIB file (.xib) via Interface Builder.

Editing a class file

Before we dive into the code, let's take a moment to look at how we find the controller we need to modify. In the Xcode Groups & Files pane, find the SDK101ViewController.h file, and click on it. The file will now be selected in the File Name pane and you can start modifying it via the code pane.

SDK101ViewController.h

The header file will contain definitions for any properties we'll need, and in this case, a couple helper methods. First, define the objects & methods in the .h file: Listing 1.

SDK101ViewController.m

Next, synthesize the Text Field & Label properties & stub the methods: Listing 2.

Also – as a best practice – it is a good idea at this time to add code that keeps memory tidy: Listing 3.

There is one more step to complete in Xcode before we dive into Interface Builder – we need to expose our View Controller so that when we run the application in the simulator – we can see it. To do that, we need to edit the existing Application Delegate.

Listing 1. *Objects & methods in the .h*

```
@interface SDK101ViewController : UIViewController
                    <UITextFieldDelegate> {
    // define text field & label
    UITextField *sdkTextField;
    UILabel *sdkLabel;
}

// define text field & label properties
@property (nonatomic, retain) IBOutlet UITextField
                    *sdkTextField;
@property (nonatomic, retain) IBOutlet UILabel
                    *sdkLabel;

// define methods to dismiss keyboard & update label
- (IBAction)textFieldDoneEditing:(id)sender;
- (void)updateLabel;

@end
```

Listing 2. *Synthesize properties & stub the methods*

```
@synthesize sdkTextField;
@synthesize sdkLabel;
```

Listing 3. *Memory tidying*

```
- (void)viewDidUnload {
    // tidy up
    self.sdkTextField = nil;
    self.sdkLabel = nil;
}

- (void)dealloc {
    // tidy up
    [sdkTextField release];
    [sdkLabel release];
     [super dealloc];
}
```

Listing 4. *Stub View Controller*

```
@interface SDKtoday101AppDelegate : NSObject
                    <UIApplicationDelegate> {
    // declare window & view controller
     UIWindow *window;
    SDK101ViewController *sdk101ViewController;
}

// declare window & view controller properties
@property (nonatomic, retain) IBOutlet UIWindow
                    *window;
@property (nonatomic, retain) IBOutlet
                    SDK101ViewController
                    *sdk101ViewController;
```

SDKtoday101AppDelegate.h

Much like in the View Controller's .h file, the app delegate .h file will define any objects we need. In our case, the goal is to expose the view Controller, so we need to update this file to make sure there is a definition for our View Controller to expose it. The code looks like this: Listing 4.

SDKtoday101AppDelegate.m

Finally, we will go to the implementation to add the code that exposes the view. Here is the code to synthesize the View Controller, and expose the view in the window when the application finishes launching. We also add code to dispose the view and clean up the memory when the application is ending (Listing 5).

That's it, we've stubbed everything we need before we build the UI & perform our connections in Interface Builder.

Listing 5. *App Delegate*

```
#import "SDKtoday101AppDelegate.h"
#import "SDK101ViewController.h"

@implementation SDKtoday101AppDelegate

@synthesize window;
@synthesize sdk101ViewController;

- (void)applicationDidFinishLaunching:(UIApplication
                    *)application {
    // expose view
    [window addSubview:sdk101ViewController.view];
    [window makeKeyAndVisible];
}

- (void)dealloc {
    // tidy up
    [sdk101ViewController release];
     [window release];
     [super dealloc];
}

@end
```

Listing 6. *IBOutlet Stubs*

```
// define text field & label properties
@property (nonatomic, retain) IBOutlet UITextField
                    *sdkTextField;
@property (nonatomic, retain) IBOutlet UILabel
                    *sdkLabel;
```

Connect It

Now that we have defined anchors for the controls, we can wire them up with Interface Builder (IB). When we defined the properties/methods, we used a special identifier that tells Interface Builder we want to be able to connect them to controls we drag and drop in the interface. Here is the key code to re-enforce how this works: Listing 6.

Notice the keyword IBOutlet – that is the magic. If you remove it – connections Interface Builder can get confusing in rapid fashion. It is difficult to connect something via the IB Connections Inspector if it doesn't exist to connect to!

Another important note – when coding, before you go into Interface Builder, it is critical to build the project. This will help with two things:

- It will ensure all your files are saved to that Interface Builder can find the control definitions
- It will help you ward off any typing mistakes or missing punctuation before you proceed

This way – when you're done in IB – you're ready to build & run.

SDK101ViewController

To get started in Interface Builder simply double click the SDK101ViewController.xib file.

Add the UI Controls

In Interface Builder add the Text Field & Label by dragging them from the library window to the view.

Connect the UI Controls

To make a connection in IB, simply drag from the Connections Inspector outlet to the appropriate control. First, set focus on the File's Owner then press command + 2 to bring up the connections inspector. In our case, we have two outlets to connect, a Text Field & Label. Have a look at the screenshot, below, what you should note is the IBOutlets you defined earlier, then the drag/drop to the control: Figure 1.

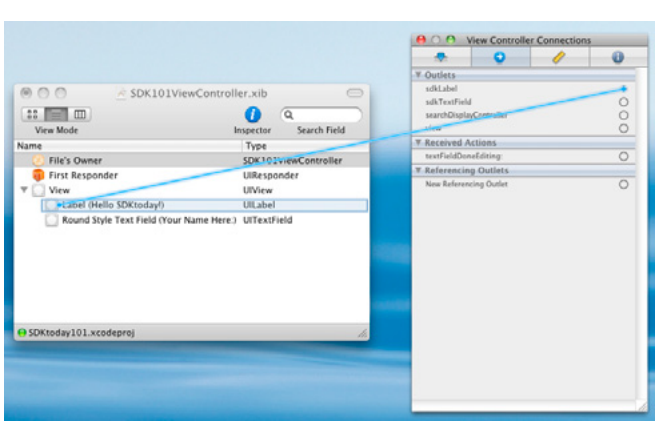


Figure 1. *IBOutlet Connection*

The final step is to wire up the new textFieldDoneEditing IB method to the Did End On Exit event for the Text Field control. This is slightly different than the IBOutlet because in this situation we are mapping an IBAction. This method will release the keyboard when we're done typing! To make this connection, do the following:

- Highlight File's Owner
- Find the action placeholder for textFieldDoneEditing
- Drag & Release it on the UITextField that will be rendering the keyboard for user input
- A pop up dialog will appear with the keyboard outlet options, choose Did End On Exit

The following screenshot illustrates what this will look like: Figure 2.

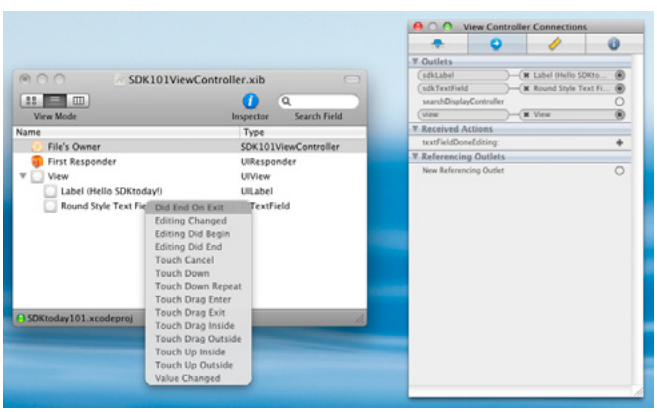


Figure 2. *IBAction Connection*

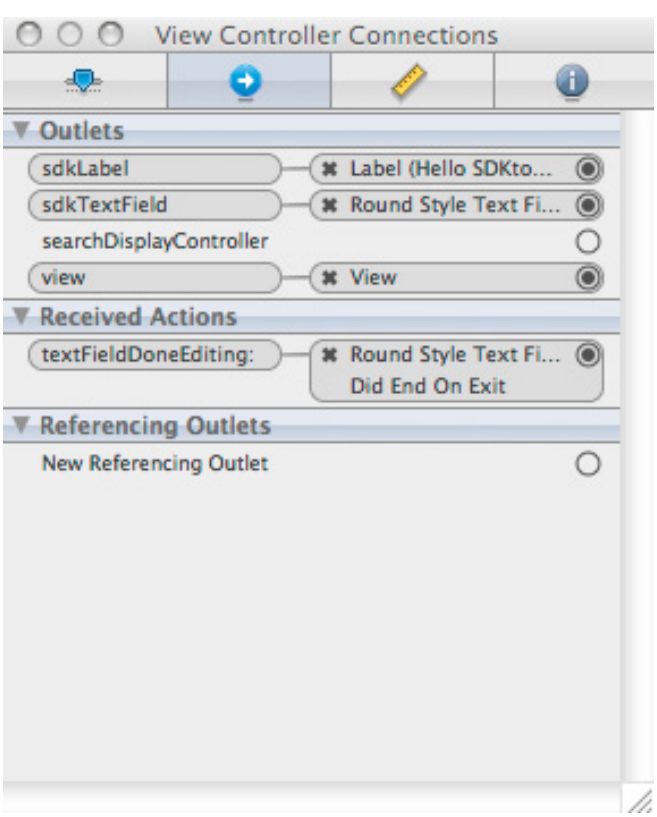


Figure 3. *Connections Inspector*

When you are complete, the Connections Inspector should look like this: Figure 3.

MainWindow

At this point, lets recall briefly when we started this adventure, we chose a windows based application, and we got something called MainWindow.xib as a result. This xib contains our App Delegate, which is responsible for rendering our application, so, next, we need to connect our view controller to the app delegate.

There are a few steps we need to perform to complete this final connection:

- Add a View Controller
- Change the Class of the View Controller to pont to the View Controller we created in Xcode
- Connect the View Controller to the App Delegate

Add the Controller

When adding the View Controller to the window, the process is as simple as when you added the UILabel in the previous section. The first step remains the same, we locate the Control in the Library pane (Cocoa Touch – Controllers), this time we are looking for View Controller. Select the control and drag it to the MainWindow.xib window, and drop it immediately under the Window item.

The Figure 4 illustrates what the end result will look like (of course you still have a few more steps to perform before reaching this point):

Notice that we updated the View Controller reference to ‚SDK101ViewController'. The steps to do this are defined below:

Change View Controller Class

- Highlight the row View Controller
- Press Command + 4 to bring up the Identity Inspector

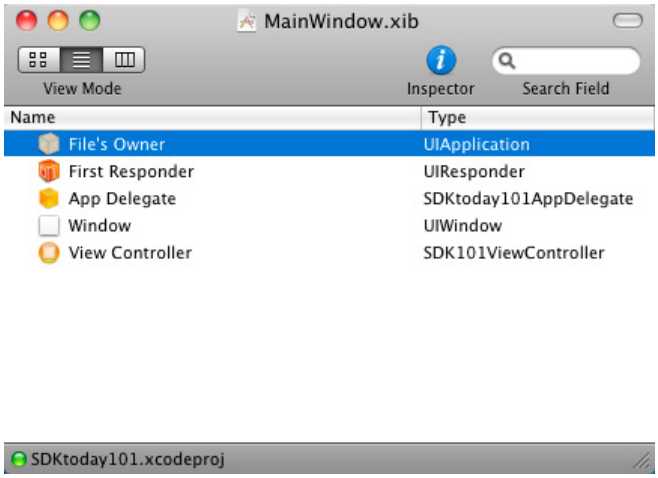


Figure 4. *MainWindow*

- In the Class dropdown list, find & select SDK101ViewController

Connect the Controller

Finally, we need to connect our View Controller to the App Delegate so that when the application loads, we can see our view!

Highlight the row App Delegate

- Press Command + 2 to bring up the Connections Inspector
- Find sdkView101Controller in Outlets & drag it to the View Controller row in MainWindow.xib

Save everything, then Build & Run your project to see it in action!

You should see that your application loads – and looks much better than before – but still doesn't really do very much. Well, that is not entirely true – it let's you type in the Text Field with a magical keyboard – but the keyboard feels less magical when you realize you can't seem to get rid of it. There was also the matter of updating that label. Don't sweat it; we are just a handful of lines away from crossing the finish line via the Complete It section!

Complete It

As you may recall, in Stub It! we added the method declaration to hide our keyboard after the user is finished typing, and to display the result to the user. Now we need to implement the logic to make that happen. In this section we will:

- Add code to enable trapping the keyboard done button
- Complete the contents of our shell updateLabel method to write some text to the label

SDK101ViewController

All of our methods are located on the View Controller, so navigate to SDK101ViewController.m and we will complete the code.

textFieldDoneEditing

Here we will complete the method that we stubbed to capture the Text Field Keyboard Done button tap event. This code hides/dismisses the keyboard (Listing 7).

Listing 7. *Hide/Dismiss Keyboard*

```
- (IBAction)textFieldDoneEditing:(id)sender {
    // hide keyboard on done click
     [sender resignFirstResponder];
    // update the label
    [self updateLabel];
}
```

Listing 8. *Update Label*

```
- (void)updateLabel {
    // get input
    NSString *nameString = sdkTextField.text;
    // check if any input was provided
    if ([nameString length] == 0) {
        // no input - default name
        nameString = @"SDKtoday";
    }
    // build message string for label
    NSString *messageString = [[NSString alloc] initWithFormat:@"101 \n%@!", nameString];
    NSLog(@"SDK101View.updateLabel - set messageString: %@", messageString);
    self.view.backgroundColor = [UIColor colorWithRed:0.000 green:0.060 blue:0.100 alpha:1.0];
    // set label text to message string
    sdkLabel.text = messageString;
    // notice we tidy up message string - but not nameString
    // this is because they were initialized differently
    [messageString release];

}
```

On the Web

- *http://sdktoday.com/* – SDKtoday App Website
- *http://kengin.ca/* – Jacob's Corporate Consulting Website
- *http://kallergisconsulting.com/* – Demetree's Corporate Consulting Website

updateLabel

Here we build the message string by concatenating a string message with the text entered in the Text Field. Notice how we formatted the string for display, and then used it to update the Label (Listing 8).

That's a Wrap!

That wraps up the Complete It section, and the step-by-step example for the S2C2 app development methodology.

Summary

This is a very simple introduction to the S2C2 app development methodology. For additional app examples, video tutorials, and full source code, check out the SDKtoday app in iTunes. SDKtoday was written in collaboration with Dennis Volovic of Dedovo Inc.

Once you master these topics and are ready to consider taking your apps to the next level with application server integration – you can check out another project I am collaborating on with Demetree Kallergis. The project/app are called Server also available in the iTunes app store by searching Kallergis Consulting

JACOB KENNEDY

Jacob Kennedy has 15+ years experience in the IT field ranging from large enterprise to individual consumer clients. He is currently working as an enterprise CRM professional services consultant by day, and entrepreneurial developer on various passion projects during evenings and weekends.

Build Better Apps

by Leveraging the Power of BaaS

Backend-as-a-Service (BaaS) providers like, Parse and StackMob let app developers to focus on building great apps by taking care of much of the plumbing that would typically be required to build Internet-enabled mobile apps.

When setting out to develop an iPhone app, you might think your time will be spent writing Objective-C and mastering the ins and outs of Apple's Cocoa Touch framework. While you will undoubtedly find yourself wading through the jungles of Objective-C, more often than not developing your iPhone app will require a significant amount of work developing the back-end infrastructure to support the app. Lucky for you over the past years a new breed of cloud service providers, called Backend-as-a-Service (BaaS) has emerged which provides for you a turnkey back-end for your app that reduces much of the pain and work that usually associated with creating an web enabled iPhone app. In the end, app developers can save time and effort that they would have spent developing what is a largely commodity piece of logic by adopting a BaaS solution.

Imagine you are building the latest and greatest photo sharing app (think Instagram), the iPhone app is only one part of a web based system to bring the app to life. You will need some sort of cloud *accessible* database to store all of the photos and app users in, on top of that, you will need also a web service layer that will be used by your iPhone app to create and view photos from the cloud database. Finally, inside of your app you are going to need the logic to synchronize data, which will be stored locally on the phone (usually through a SQLite database), with the data being created in the cloud by all the other users of the app.

If you decide to go it alone and build all of this plumbing yourself, expect this work to create the back-end web service and in-app synchronization logic to eat up a good chunk of your development time. In our first app, Bahndr, we spent a solid three months of development developing a back-end web service that sat on top of a SQL Server instance, in addition to the glue code written in Objective-C that pushed and pulled data from the service. The actual UI constructs that made up the visible app was relatively simple and required four weeks to complete. Looking back, if we had known about Pulse and StackMob, we could have shaved months of our development time by adopting an *out-of-box* BaaS solution that would obviate our need to roll our own.

To all those looking to jump into the app space, take a good, hard look at BaaS offerings from Parse (which was recently acquired by Facebook), StackMob and Kinvey before you decide to reinvent the back-end wheel. Each of these BaaS offerings provide a similar set of capabilities that include:

- a cloud hosted database that supports a user-defined schema,
- a REST based web service layer to pull and push entities to the database through,
- a generic push notification event handler,
- a callout to user supplied logic (usually written in JAVA or JavaScript) which will be executed on the server for additional post-processing of data changes made through the web services layer,
- a iOS SDK that replicates the databases schema atop of the phone's CoreData provider and which automatically takes care of syncing to and from the web service objects.

Most BaaS providers charge for the use of their services on a per API call basis. While this might sound expensive, it actually isn't. For most BaaS providers, you only end up paying for the service once your app achieves some popularity, which means you can de-

velop, test and launch your app atop a BaaS solution with little to monetary cost!

BaaS solutions do come with risks, mainly that you are now introducing a third party dependency to your app. Bugs in the BaaS platform can end up stalling your development, and downtime of the hosted database can end up completely crippling your app. Then again, any bug in your own application or network connectivity will also have the same effects. These downsides should be weighed next to the potential development benefits of adopting a BaaS solution.

Looking back, if we were setting out to develop our first iPhone app again, we would most likely choose a BaaS solution like Parse, in lieu of creating our own. BaaS offerings let developers focus on designing and developing great app experiences without worrying about the plumbing that connects it to the web.

BOBBY GILL

Bobby Gill is the founder of Blue Label Labs, a mobile development lab based in New York and Seattle. Bobby is also the author of "Appsters: A Beginner's Guide to App Entrepreneurship" and the Editor of Idea to Appster, the trusted source for news, articles and tips to help startups and entrepreneurs build better mobile apps. Bobby graduated with an MBA from the Columbia Business School in 2011. Prior to Columbia, Bobby spent four years at Microsoft, as a Program Manager within the Forefront Identity Manager (FIM) product group. During that time, Bobby served as an engineering lead for the FIM product with a specialization in database and web service design. Bobby graduated from the University of Waterloo, in 2005, with a Bachelor of Mathematics specializing in Computer Science.

Power of Push

Push messages are one of the most instant, limitless and personal ways for a brand to communicate with its customers. Every customer who uses your brand's app can accept (opt-in) to receive push messages. As long as your customers carry their smartphone or tablet with them, they can be reached anytime and anywhere. An ideal channel to activate, inform or service your mobile customer.

Marketers have several reasons to adopt push functionality in their app, it gives them a low cost, instant and personal communication channel to their customer base. The direct effects of push are easy to measure, but the indirect effects are just as interesting. Push messages generate traffic to the app, giving the brand the opportunity to expose the app user to an unlimited amount of the latest media or information.

In the app they can read the information, buy, share or give feedback. It increases the relevance of an app for the user, and ultimately increases brand awareness. The frequent use of push is known to increase the frequency and length of engagement of app users. Compared to SMS, push does not make use of the telecom network. Recipients just need a connection to the internet. This makes the service highly scalable and cost effective. As a result, Service2Media is able to offer unlimited push messages with consistent monthly fees.

What can push do for your brand?

The secret behind the popularity of push is very likely to do with control residing with the user. With only one or two clicks recipients can decide to opt-out from receiving messages and, if they want, opt-in again later.

Segmentation and analytic tools help brands to determine which type of push message will be most relevant for the recipient. In this way the push service can help brands to determine what type of message is most relevant and what frequency is most optimal for each user.

An app combined with a personal account, login, demographic information and category-specific opt-in enables marketers to target different segments. Segmentation can be based on the customer's (personal) interests, or demographics such as gender, age and location.

Analytics that measure the effect of a specific push message is very precise, providing data on the number of screens visited, how much time users spent with the app, and enabling the monitoring of generated traffic and the development of random A/B splits.

Push vs. SMS vs. Email?

Do you already have an SMS service or do you prefer using email campaigns to reach your audience? Both can be used as supplement or alternative to push messaging. No every push user is in your email or SMS database.

Push messages have very strong opening rates, are less cluttered and have an instant delivery level like SMS, but with the media-richness of email. Next to that there are differences in user behavoir and technique when comparing sms and Email tot push.

Furthermore, push messages give marketers the ability to channel recipients. SMS recipients often get lost

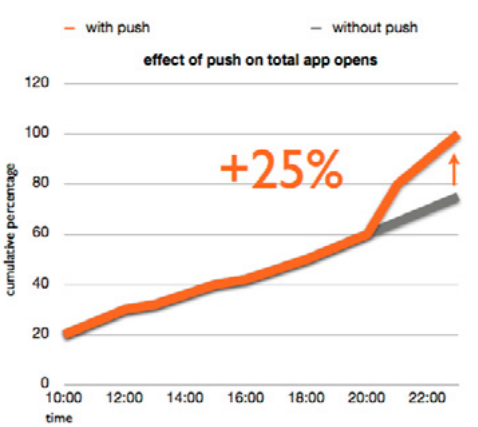


Figure 1. *Effect of mass push messaging on total app opens*

or are redirected to a mobile website. A push message will redirect the user to specific content in the app and track their behavior.

Push makes your app more relevant

The graph on top shows the cumulative percentage of app opens within 24 hours, and the uplifting effect (on average 25% increase) that a push message has on the app's total daily visits (Figure 1). This implies that on a daily basis you can activate at least 25% of your customer base and efficiently channel them towards specific content in your app.

How fast are instant push notifications?

Push messages arrive instantly, with around nine percent of the daily opens occurring during the first minute of the push message being sent. Push messaging is a highly scalable

approach to alerting customers. One message can be sent to millions of users at the same time, or different versions of messages can be sent to different groups at the same time. Almost 19 percent of the daily opens occur within 5 minutes of the push messages being sent. It is assumed that these push messages were allowed to trigger the smartphone to vibrate, light up the screen or provide an audio signal.

iPhone, Android, BlackBerry and Windows Phone have different ways of displaying push messages to their users.

Some operating systems allow for are more invasive and alerting push notifications than others. However, just as with other app users' details, it is possible to adapt your message's content, timing and priority taking the OS type into consideration when sending out either personal, grouped or mass messages.

Why choose for a push framework?

Instead of using several back-ends, frameworks and interfaces for the different apps and OSs you want to use push notifications for, it saves time and money to do it from one framework, with one back-end and one interface environment. All statistics can be combined to enable analytics to track and monitor your push messages.

Push messages can be sent for every OS to any type of iPhone, Android, BlackBerry and Windows Phone device. Push messaging is secure, and easy for average developers to integrate quickly into a new or existing app.

Show me, what's the concept?

Two main actors play a role in the push messaging process: the push initator and the app instance. The push initator sends the messages. The app instance receives the messages. In between, M2Active push and the vendor push networks operate. The following basic mechanism is used to send out a push message: Figure 3 and Table 1.

How do i build an app that receives push messages?

Including the capability of receiving push messages in your app is not rocket science. If you are already using the M2Active platform, you have a convenient client API available for Push. If you want to include push in your native app, you can easily use the REST API we have defined for you, including methods for:

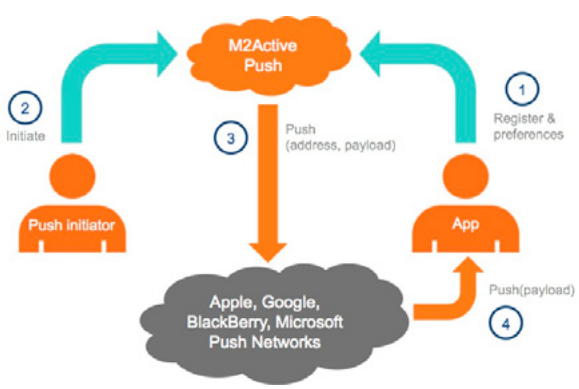


Figure 3. *Push message from initiator to app user*

Figure 2. *Example of push notifications on iPhone5*

- Requesting a push address
- Registration and deregistration of a mobile application instance
- Registration and deregistration of user preferences of such an instance.

The following figure shows the main operational data flows depicting how an instance of a mobile application can register with M2Active Push and how the push invocator side of that application can subsequently send push messages to such an instance (Table 2).

If you prefer to use the API instead of the portal, several REST web services are offered, including:

Application register
This web service allows an app provider to register the application's credentials with the M2Active Push server. These credentials will be used for identifying push requests to the various push networks. There are five different versions of this service to cover the various vendor push networks.

Query
Using the web Query service an app provider can obtain the number of devices that match a certain push criterium.

Wipe
The Wipe web service allows an app provider to remove a mobile application instance's push address or preferences from the M2Active Push database. This web service can, for instance, be used to disable sending push messages to a specific user.

Table 1. *Flow of Push Messages*

Flow	Name	Explanation
1	Register and and preferences	The client app instance registers itself at M2Active Push as a legitimate destination of push messages, indicating from which server app it wishes to receive messages. In addition, the app instance can indicate its push preferences to M2Active. These preferences will act as filters for incoming push messages. Interfacing happens via a REST API over HTTPS.
2	Initiate	The push initiator invokes the push message to be sent. This can be a human being using the M2Active Push portal, or another backend system using the REST API over HTTPS. Of course we provide a mechanism to identify this push initiator to be registered as a legitimate source. We will authenticate the connecting server by checking its credentials and we will deny access from unregistered IP addresses.
3	Push	When the Push process is invoked by the initiator, the message is sent to the push providers. M2Push interfaces with Apple (APNS), Google (both GCM and C2DM), BlackBerry Push and Microsoft (WP7, WP8 and Windows 8 (WNS, MPS)). The message will only be sent if it passes all filter criteria set up in the filters.
4	Deliver	The messages will be delivered to the devices running the client app instances via the various supported push networks.

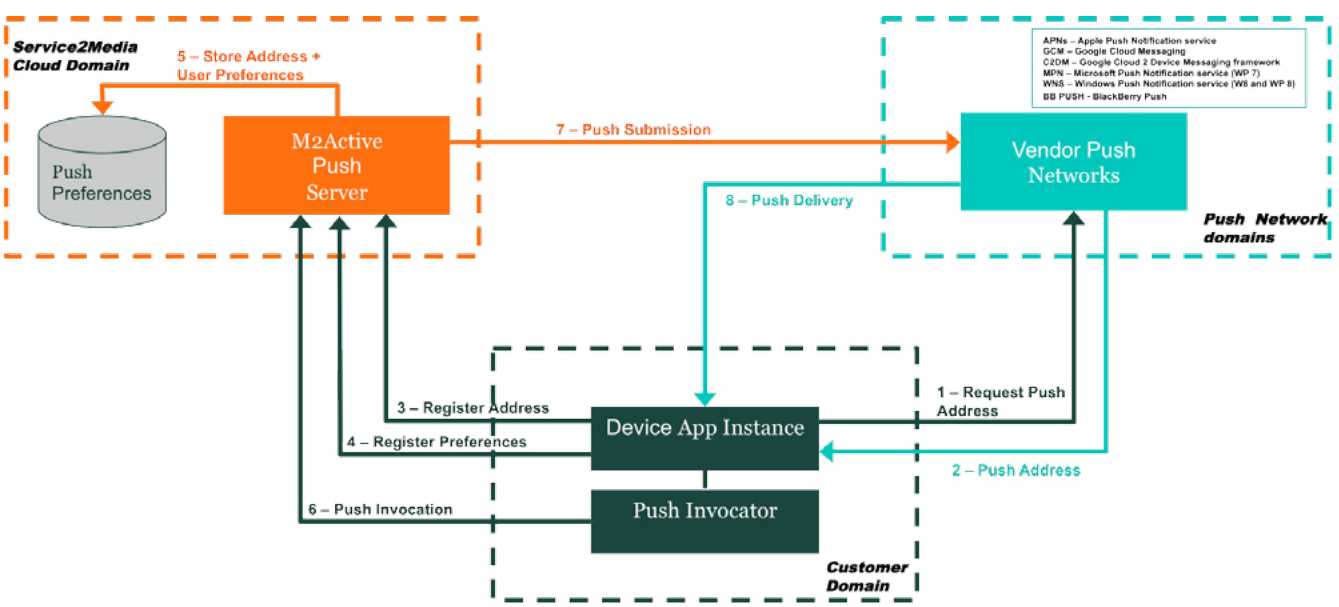


Figure 4. *Procedure M2Active Push*

Push Invocation

The push web service provides the app provider with an interface to send a message with accompanying payload and push profile to the M2Active Push server, which will take care of determining which device to send the message to and subsequently issue pussh submission requests to the appropriate vendor push networks.

Preferences and Preference Profiles

Messages are only delivered to the client if they pass the M2Active Push filtering mechanism. This filtering mechanism matches the preferences of the user (the Client Push Preferences) with the target group for the message, selected at the server (the Preference Profiles). This 'Everybody happy' filter makes sure that users only receive what they want to receive and that the app provider can carefully select his target audience (Figure 5).

Client push preferences

An app instance that intends to receive push messages can expose a list of keywords (preferences) that provides clues as to which push messages it is interested in and which not. These preferences are matched with the preference profiles you set on the M2Active Push server. The app is free to declare any number of preferences. Each preference is given a unique name and declared with one of the preference types BOOLEAN, INTEGER, STRING, DATE or TIME.

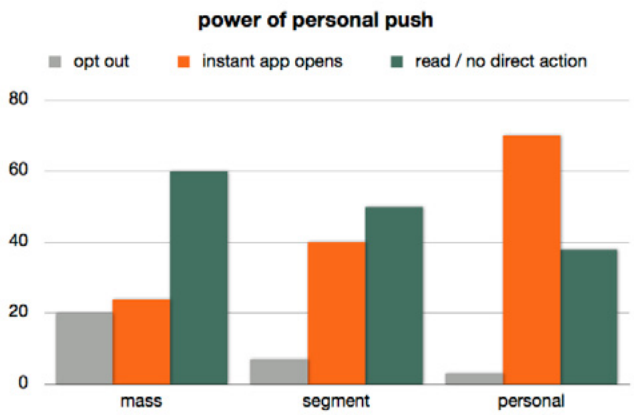


Figure 5. *Power of Personal Push*

Table 2. *Procedure M2Active Push*

Flow	Name	Explanation
1	Address	An instance of the mobile application (i.e. the mobile device running the app), requests a unique push address from the vendor push network the device belongs to. In doing so, the device indicates that it intends to receive push messages for the said application. This request must be sent when the app is started for the first time. For practical reasons, it is recommended to send this request each time the application is started on the device, since certain push addresses have a limited lifetime.
2	Push address	The vendor push network responds by sending a push address to the device. This address is a unique reference for the application instance (combination of device and application) and is used by the relevant push network as a delivery address for the said application instance. The push address maps directly onto its native push network equivalent. For instance Apple uses a 'token', Google an 'identifier' and Microsoft a 'channel'. The format and the semantics of the push address also differ between the various networks, but this is not an issue since it is handled by M2A Push as an opaque string.
3	Register address	The mobile application instance registers its intent to receive push messages with the M2A Push server by sending it its push address.
4	Register user preferences	The mobile application can optionally send a set of user preferences, which it expects the M2A Push server to use as criteria for filtering push messages before sending out to the push network. Note that all user preferences are defined at application level, based on a predefined set of preference types.
5	Store address + user preferences	The M2A Push server stores the push address and associated user preferences for future use. All actors involved in the submission, delivery and receipt of push messages are now properly initiated and the end-to-end push service may commence.
6	Push invocation	The push Invocator takes the initiative to send a push invocation to the M2Active push server. The push invocation includes the payload of the push message, as well as an optional preference profile. The preference profile determines which subset of users the message is targeted at.
7	Push	The M2A Push server matches the message profile with the preferences of the individual users and thus selects which users to send the message to. In addition, it determines which push network each individual selected user is on. Finally, it submits push message requests to each of the push networks involved, as appropriate. Note that in a typical use case, a single received push invocation will be expanded to multiple push submissions.
8	Push delivery	The proprietary push network delivers the push message to the individual device, based on the destination information provided in the push Address.

The preferences are registered at the push server by sending a list of push preference 3-tuples, where each 3-tuple contains the key, the type and the value of the preference, e.g.:

```
"weaterupdate", BOOLEAN, true
"age", INTEGER, 35
"city", STRING, "Amsterdam"
"category", STRING, "paid user"
"laststart", DATE, "now"
"nextmessage", DATE, "tomorrow"
"birthday", DATE, "September 13 1975"
"startofworkingday", TIME, "08.00"
```

Preference profiles

Each push invocation can be accompanied by a preference profile that is matched with the user preferences of each app instance that is within the application's domain. Using the preference profile, push messages can be submitted to selective devices, e.g. all devices in city 'Amsterdam', all users in the category 'Paid User' or indeed all devices in "Amsterdam" of category "Paid User".

The preference profile consists of a logical expression where keys and associated values are used as operands. Examples of expressions are:

```
(City == "Amsterdam")
(Category == "Paid User")
((City == "Amsterdam") && (Category == "Paid User"))
((weatherupdate == TRUE) || (city == "Amsterdam") ||
                    (birthday < [january 1 1980]))
(age < 35)
((age > 40) || (weatherupdate == TRUE) && (city ==
                    "Amsterdam"))
```

If you want to know the exact details of the filtering processes, you can download the Getting started guide here <link>. If you want to try yourself, get a two month free trial at <link>.

Figure 6. *Example of personal push messages send*

Security

It is important to keep your push messaging safe. We have therefore added several strong security mechanisms to prevent unauthorised use of your service:

Both client applications and push invocators are authenticated by means of username/password credentials. These credentials are issued by Service2Media to the app provider via an 'out-of- band' side-channel (typically email) as part of the application registration process,

Confidentiality is achieved by communicating with the M2Active Push server exclusively over secure channels based on SSL Encryption over HTTP (HTTPS),

Impersonation of push invocators is prevented by IP address filtering. As part of the registration process, the app provider's IP address range can be configured on the M2Active Push server. Incoming requests from remote hosts outside of this range will not be accepted,

Certificate message signing (Apple). Applications can upload their Apple push provider certificate such that the M2Active Push server can be authenticated by the APNs as acting on behalf of the app provider (two-way SSL authentication).

Privacy is of major importance for us as well. We do not store personal data. If you want to filter and send messages to individual named users, you will have to setup a database yourself that links personal information to device IDs. We provide hooks to do so. In this way, we function as a proxy only and we are not storing any personal data. Even better, your personal data is in your domain and possession. And that is the best place to keep it.

SERVICE2MEDIA

If you want to know the exact details of the involved processes, you can download the Getting started guide from www.service2media.com. If you want to try yourself, get a two month free trial at http://www.service2media.com/push-test-drive/.

nicky in the clouds

scalable creativity

Get your Projects Off the Ground !

www.nickyintheclouds.com/sdj2013

LIKE US ON facebook

Interview with

Doug Panchyshyn

Doug Panchyshyn is an independent computer consultant who started developing in the C language back in the 80's and has launched into other languages and platforms with a focus on C. His current focus is on Objective-C for iOS and OS-X. He is also a UNIX/Linux web-server, database, and systems administrator. He has developed for both small and large organizations including many manufacturing companies designing and supporting ERP and process control, and has developed his own Relational Records Management solution for SoIntuitive.com. He develops for all major platforms while working for organizations such as IBM, AT&T Canada, Rogers Enterprises, Manulife Financial, Nortel Networks, the Ontario Government, the Canadian Institute for Health Information and others. Software Developers Journal asked him some questions regarding iPhone development.

What do you need to be an Objective-C developer?

I see this question as having a very specific focus on the word "you" and on the language "Objective-C". It is less about the tools that the individual needs and more about how much time an individual or a company has already spent writing software, and how far they want to take their Objective-C development. There are some myths about Objective-C that exist in the minds of many. I'll cover these in my first article which should help in further answering this question.

Do you and how do you divide development members into multidisciplinary teams? (front-end, backend, UX, etc ...)

How I divide a development team depends much on the size and length of time determined for the project. It also depends on the number of resources and skills my team members have. I see my project as large in size only if the estimated time for delivery is high. I begin by looking to see if a project includes server side requirements, if it needs to handle thousands of simultaneous users, if it needs to have fail safe ability, and if it needs to be a fault tolerant solution. I would certainly then break down team members into their respective strengths along with identifying individuals whose skills can cross over into the skills of another member and thus complement the other as each need arises. On the other hand, if my project is relatively small in size and the time estimate is low, I may even go right down to the extent of using a single skilled developer who has multiple disciplines and who documents their code very well. I would put preference on an individual that can also go beyond just writing code, as for instance, one who is also very artistic with vector graphics and is a stickler for quality. Coming from a manufacturing background, I first "divide and conquer" and then "concur". Multidisciplinary is, in a way, similar to multi-tasking when trying to get an individual to do "fast switching"; it's an extremely complex thing to do and it depends much on your environment and what you have to work with. However, when it comes to an individual versus a team, managers are the ones that really need to have the so called "fast switching" capability. Managers need the discipline along with a broad set of cross system experience. In the end, once you bring individual team members together in the right way with their individual strengths, each of their disciplinary strengths perpetrates across the entire team. This turns into wisdom or knowledge applied.

What is your opinion concerning the need to use only an Apple product to develop for iOS devices? Do you think it limits the market? What are the pros and cons?

I look at "opinions" as the fuzzy logic of our real world. What makes or breaks a market is very subjective to

many details. On the other hand, I'm sure the way Apple sees things, their opinion, is similar to how some developers see things. For instance, I'm sure Apple feels that if the developer is going to need an iOS device to test their app on and the user of your app is also going to be using an Apple iOS device such as iPhone, iPad or iPod Touch anyway, why on earth wouldn't you use an Apple product to build it? I wouldn't say that I have a strong opinion but I do tend to lean a certain way due to several concerns. One concern involves more business logic than it does technological concern, since in the end, it really amounts to being Apple's call to close the door on API's and other technologies that they do not approve. I often ask the question that if Apple suddenly chooses to say they won't allow a certain API that is part of your binary deliverable to the App Store, then what will you do? You could end up with a whole lot of code on your face! At the very least you could have a pretty big delay in being able to get your app out before resolutions are made between Apple and the third party API or tool.

The second is related to licensing agreements. If Apple says not to install their operating system on non-Apple hardware, then I find that pretty simple to say no to. If an individual developer touts the fact that they broke Apple's agreements and is able to develop an iOS app on a Windows machine, then I ask myself if I should trust them with any of my agreements.

A third is a matter of the inexplicit and a "fear factor" that unclear agreements can stir up. Questions such as can this turn into ethical or even legal concerns? What does Apple's complex agreement really say? Is my company large enough and are we prepared to spend money on resources outside the realm of the technical side, namely the legal side of software development? I've read so many discussions on professional forums where questions similar to these cannot be clearly answered and only fuzzy, grey opinions float around leaving people wondering what they can do, can't do and whether they should do. It is said that grey is an intermediate color between black and white, meaning literally a color "without color". What a great toy this can be for lawyers to play with. To me, grey, equates to cost. To add to this, having come from a manufacturing background, I much prefer to break things down into smaller components and processes rather than trying to stuff everything into one mid-sized smelting pot or one small manufacturing plant. Instead, I learned to develop for Windows, Linux, Mac and iOS for both client and server side. I do my code writing, testing, etc., on the operating system and platform that the manufacturer recommends so that I can get finer grained technical support and more tightly coupled business relationships. When I developed for FreeScale micro-controllers, I used what FreeScale recommended, Windows and their recommended IDE. I originally learned C, and therefore, look to platforms that let me extend my skills that I've developed over the years in C. The way I see it is, C is my platform of choice. More "fuzzy logic" or "opinion" to add to this in the form of a question: if Apple-platform-developers themselves need to develop one product release for the Mac, another for the iPhone and yet another slightly different version for the iPad, then don't I need to ask myself how dependable or limited other systems will turn out to be? As for the question, "Does this limit the market of your app?" No. If you have the right team and the right knowledge and the right timely delivery approach, it shouldn't limit the marketing of your app. But again, coming from a manufacturing background, my "opinions" are formed by my experience in first considering the risks, then breaking things down into parts and processes. I then look at the skills of my team and external resources, and then ask: "What are my ethical obligations?", "What are my legal obligations?", "If I need to test for market interest, and there isn't yet a proven demand for my idea, what do I feel comfortable spending to first test the market for my idea?". Finally, why shouldn't I try to deliver on one platform first. I think then, the pros and cons of using cross-platform tools and services, should sift to their proper level.

Objective-C is an Object Oriented programming language which was released in 1992. Currently it's Apple's preferred language to be used when developing MacOSX and iOS. How has this technology been adapted to current mobile devices?

Objective-C is a true extension of the C language. Anything that was written in the C language in the 70's should still be able to be compiled and run using the Objective-C language and compiler. Prior to the mobile market, Apple had a lot of experience providing Objective-C tools for the original Mac market in the 90's, then afterward, for the Mac Intel based machines in the 2000's including OSX. Cocoa libraries were borrowed and modified from the Mac version to create Cocoa Touch for the mobile devices. Apple originally did not make an Objective-C development environment available for developers on the initial release of the iPhone. Apple did release some guidelines for creating client side web apps that took advantage of certain features of the iPhone environment. I was on top of it the moment I got my hands on their mobile device. Then in late 2007 Apple announced that an iPhone SDK would become available in early 2008 to be used with their Xcode IDE. It was one of the most exiting times when Apple offered to advertise apps that developers developed for the iPhone and iPod touch. I signed up and bought the iPhone SDK the moment it became available in Canada and have been focused on it ever since.

How do you adapt iOS apps to run on different mobile devices on the market? Is it very expensive? Do you need to have different development teams for each?

The term "different systems" is a very broad area to cover and can include external attached devices. However, to me the term "different systems" means different mobile devices such Android, Blackberry, Microsoft and others. I'll first address the question, "Is it more expensive?" Put simply, no matter how many platforms you desire to develop across, it is always going to be more expensive. This is due to the fact that each mobile device works quite a bit differently from the other. Additionally, each version of operating system for each device can be considerably different and this makes getting an app to work on different versions more complex. I have had companies contact me asking if I can rewrite apps so that they can work across all mobile devices and versions of operating system software by using a single development platform. According to my finding so far, that depends! Are you using a lot of geometry? Are you developing a business application with very simple database requirements? Do you want to use Apples iCloud. Do you want to use AWS or some other cloud service. Are the libraries available for the services you need to use available across all devices and versions? Will you run into any potential issues with licensing agreements and requirements that Apple or others puts on developing apps for the App Store other devices? Is the app going to be an Enterprise App where the development license is $299 so that you can host your app on your own App Server? If an Enterprise App, you may not need to conform to the same restrictions as the Apple App Store. The list does get extensive, and unfortunately, somewhat expensive. As for the question: "Do you have different development teams for each?", To date, I have not yet seen a cross development team use a cross platform tool where the app and its code is somewhat complex and the same code works consistently across many devices, not to mention, all versions.

Since different systems can also refers to connected devices and server side solutions or other systems, I would summarize as follows: different systems can include smaller devices that connect physically via the older style 30-pin connector and now the newer 9 pin "Lightning Connector" allowing it to doc to applications via iTunes on Mac and Windows, as well as communicate with other devices. Some have also used their ingenuity to make scanning and detection systems that work with the iOS devices audio port. There are also wireless technologies built in to adapt to Bluetooth systems, as well as WIFI capability to adapt to server side technologies. In my experience, having seen a lot of companies simply adapt to existing server side systems as an afterthought, I have found that in the end these afterthought systems end up being very slow and cumbersome. The result of this can be very expensive, when these get rejected by users. Expense comes from needing different developers for each system that your app connects to. On the other hand, developing systems from the ground up to take advantage of mobile devices, can make the user experience much better. Having different development teams is a must when the sizes of the systems being adapted or connected too are many and/or vast.

At a company's level, what do you think of this "bubble" of newly created companies developing apps?

The way I see it, software development has not really changed all that much. Although more experience has been gained as time has passed, software development has become easier in some respects to use, along with knowledge of how to do things. However, how to not-deliver-junk, is just as important as it has ever been, and even more so as systems become more complex when interacting with other systems in order to develop high quality apps. It's the high quality apps that are enjoying the longer shelf life. The larger more experienced companies with knowledgeable individuals still seem to come out as the long-term winners. Some lucky few who do not have a lot of software development experience have come out with some apps that were short-term winners and few of the lucky winners are staying on top of that game.

If any knowledgeable person can learn how to make apps is this beneficial and can it affect the software development market?

There is always more room for software developers. It is beneficial for teams and their members who already have more knowledge and who can keep on top of current technological trends. The biggest benefit is still in individuals and teams who have strong visions for future software development and design of new products. The more knowledge people have the more things they will create. It will only get better but things will become more complex and specialized. It will become even more complex and specialized as both physical devices and the software that run them come together. Specialized persons have become very beneficial for both companies and this is benefiting software developers too. I firmly believe that we will soon see and hear a phenomenal bang as we fast track into a new era of manufacturing that couples itself tightly with software development and tools, broad knowledgeable persons and specialized teams. I believe we will very shortly see some very breath taking development taking place.

by Raul Mesa & SDJ Team

wibidata

Effective Use of Burn Charts for Scrum Teams

Burn Charts are an effective way to keep your team and stakeholders informed as to the team's progress. There are two variations, Burndown and Burnup, to communicate different information to interested parties as described later in this article.

By effectively utilizing and interpreting Burn Charts a team can understand existing trending, identify signs of trouble early on, forecast potential scope impacts due to team or external factors, or use them to communicate many other trends taking place in the project.

Assumptions

Readers are assumed to have basic knowledge of the Scrum methodology, specifically (capitalized terms are defined in the Glossary):

- Burn Charts,
- Timeboxing,
- Iterations,
- Information Radiators,
- User Stories,
- Story Points,
- Commitment made by team,
- Acceptance,
- Velocity.

Fulfilling a Need

The Burndown chart was created by Ken Schwaber in 2000 while working at Fidelity Investments [1] to visually represent the amount of remaining work during the Iteration. In 2002 the Burndown chart, and variations, increased in popularity among Scrum practitioners [2] with several examples described throughout this article. In practice, ScrumMasters are routinely the originators and maintainers of Burn Charts and post the charts on Information Radiators to visually and easily disseminate information without having to ask for more details by interested parties. Additional uses of Burn Charts include:

- justification for additional resources due to team velocity being hindered by either lack of human resources or tools,
- how unavailability or consistency in Product Owner (PO) direction is inhibiting team to meet their Commitments,
- motivational tool to encourage direct or indirect team members to meet or exceed Commitments in a single Iteration.

Frequency and Use of Burn Charts

In the following examples it is assumed that a single team's performance is being measured for the different Burn Charts.

Daily Burn Charts

Daily Burn Charts should be utilized if at all possible. Daily charts are useful to visually represent where the team is at a specific moment in time and to understand how the Iteration is progressing. It is important to note that with the information shared in a daily Burn Chart

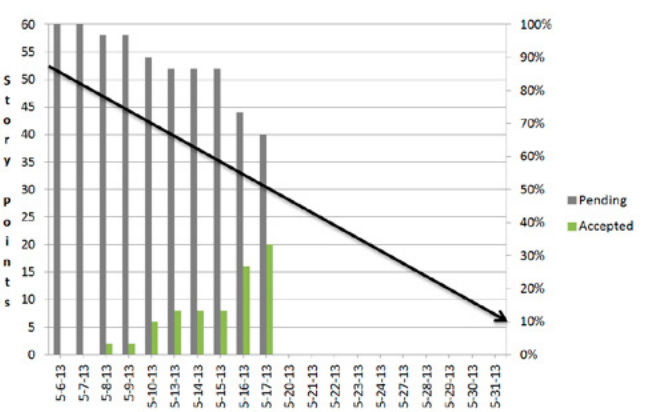


Figure 1. *Iteration Tango Burndown with Target Pending Line*

there i a considerable risk of a person misinterpreting the data. Example of a daily Burn Chart is presented in Figure 1.

Nine days into the twenty day Iteration, the trendline is clearly showing the team at risk to not deliver upon their Commitment by the end of the Iteration. This is an example how a possible misinterpretation could occur. Unless the receiver of the chart was familiar with the details of the team's Commitment, involved in recent discussions with the team, and being involved in the daily stand-ups, misinterpretation is likely to occur. Relevant data and communications may not be documented given the encouragement of team's sharing information with each other real-time. ScrumMasters need to be diligent in clearly articulating the true situation to mitigate the risk of misinterpretation as much as possible. If the team is in danger of missing their Commitment the reasons need to be documented and shared. Is the ScrumMaster having difficulty clearing an obstacle for the team? Did the team find that they consistently underestimated the amount of work it is taking to complete User Stories? If no cause for concern then the ScrumMaster needs to provide justification as well. Explaining how the team recently completed a difficult User Story that many other User Stories were dependent on and now rapid progress is being made. This level of information is important so the receivers of the information can adjust their plans accordingly if necessary.

An additional use of the daily Burn Chart is for motivational purposes. A ScrumMaster should talk with the team on their preference of Burndown vs. Burnup. Figure 1 could be used as a motivator if the team is receptive to a dash-to-the-finish type chart which is shown by the Target Pending Line going to zero at the end of the Iteration, though some may perceive this negatively since the attention of the reader is not focused on the Acceptance trend. A Burn Chart with a more positive approach, in which the User Story Acceptance % trending may be preferred as shown in Figure 2.

By changing the trendline from a downward indicator of progress to an upward one the focus of the chart has the potential to be interpreted as a more positive message for individuals that perceive progress with charts showing upward trends.

Weekly Burn Charts

Weekly Burn Charts are necessary if daily charts are not being posted. These charts can be utilized for short and long-term purposes:

- focusing on a single week of work,
- comparing week over work in the Iteration,
- multiple weeks across two or more Iterations.

When focusing on a single week of work a Burn Chart provides a very limited view of the team's progress and likely will not be of use until mid-week at the earliest. Since Commitments are based across the duration of the Iteration and not planned down to the weekly level misrepresentation easily occurs. A one week Burn Chart should be discouraged if at all possible. If weekly reports are necessary providing the following would communicate the most realistic view of the team's progress.

A week over week, in the current Iteration, Burn Chart yields more relevant information on current status than a single week view. Though there is not the level of detail as with daily Burn Charts the trends and data are the same, but caution should be urged when sharing this type of report as the results appear more exaggerated due to the lesser amount of data points as shown in Figure 3.

By comparing Figure 2 to Figure 3 one deduces that it is more favorable to post Figure 3 over Figure 2, which would represent a more positive outlook to the team and stakeholders. This is why caution is urged as this type of reporting can lead to incorrect conclusions if the supporting information is unknown to the reader. Again, the ScrumMaster must be diligent in communicating the true status of the Iteration.

A Burn Chart of multiple weeks that cross over Iterations can be generated in an effort to view how a team performs during specific weeks of Iterations, it also allows us to compare between Iterations. As discussed previously, caution should be taken when posting this data as a team does not make a Commitment based

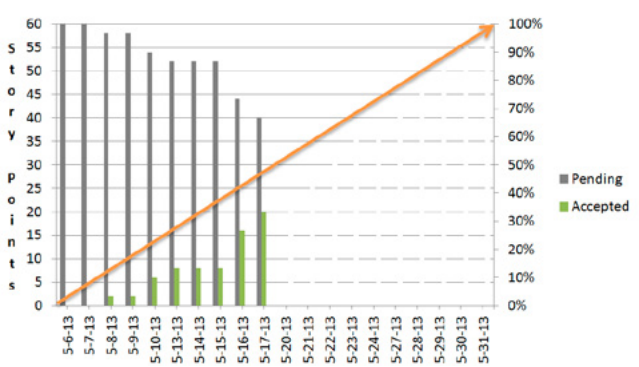


Figure 2. *Iteration Tango Burnup with Target Acceptance % Line*

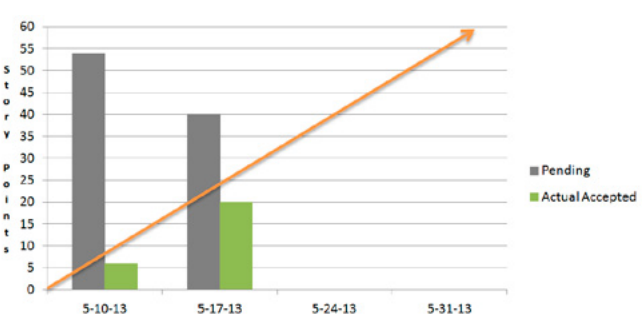


Figure 3. *Iteration Tango Burnup with Target Acceptance % Line*

upon a projection of work being completed on a weekly basis, but instead across the entire duration of the Iteration. Figure 4 represents an example of a Burn Chart covering several weeks across two Iterations. Several pieces of information can be yielded from Figure 4.

The first is the trending of Pending and Actual Accepted User Stories in regards to the individual weeks between Iterations. When comparing the same weeks across the different Iterations we can see a consistent pattern emerge. The team is completing the same amount of Story Points within five points when compared to the same time in the previous Iteration. This information is valuable for forecasting potential future performance and can assist in detecting any troubling trends.. Do not compare different weeks between Iterations as the measurement will not give an accurate gauge of performance. An example would be comparing the first week of Iteration Tango to the third week of Iteration Umbrella.

Secondly, it is possible for Velocity to be measured with the need to take into account incomplete Iterations. However, this calculation should be interpreted as anecdotal as you will need at least two full Iterations of data before a Velocity calculation can reliably be measured [3], ideally three or more should be used. Velocity measurements are of more value when reviewing Release Burn Charts as outlined later in this article.

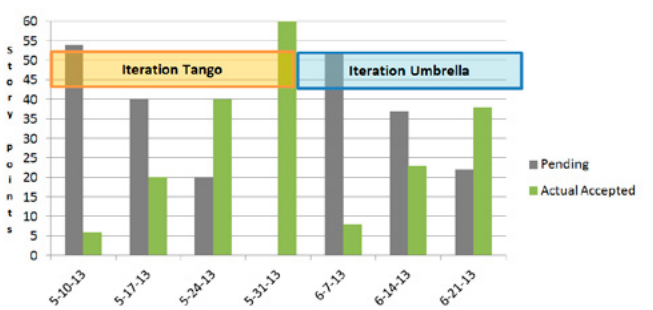


Figure 4. *Multiple Weeks Across Two Iterations*

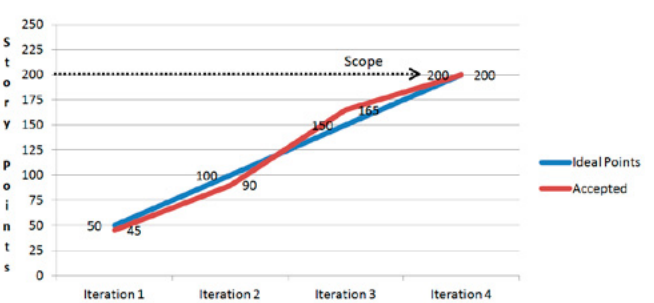


Figure 5. *Release Burnup*

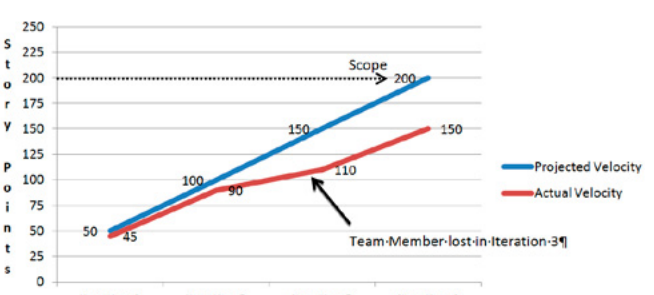


Figure 6. *Release Burnup with Loss of Team Member*

Release Burn Charts

Release Burn Charts provide insight into:

- team performance across or between Iterations,
- calculate Velocity,
- forecast potential future team performance.

Figure 5 conceptualizes a Release Burnup with four Iterations, a total of two hundred Points of scope with Points that accumulate incrementally throughout Iterations.

Team performance across Iterations is useful in that it allows us to calculate Velocity. Velocity is simply the sum of all Accepted Story Points from the previous Iteration. Figure 5 shows that for Iteration 2 the team's Velocity is 45 and as we can see the team maintained this Velocity through Iteration 2. The team's Velocity is directly tied to several factors that can cause significant fluctuations, examples being: team member participation rates, contribution levels remaining constant, consistent accessibility to PO, not dependent on expertise outside the team, trade-offs with scope changes when new scope is introduced and Iterations not terminated prematurely. It is important to note that Velocity is not an absolute measurement and is more useful when multiple Iterations worth of data is available. By taking the average of multiple Iterations a normalized number is produced and is a more accurate gauge of team performance that can be used for forecast purposes.as the team member is on the project longer they become more effecint their knowledge becomes more extensive

Being able to forecast future team performance is crucial as changes will occur throughout the project's life. Forecasts are necessary to evaluate: potential scope impacts due to the adding of team members, reduction in team members due to attrition or other factors, scope increase or decrease to name a few. At the beginning of the project, a ScrumMaster should evaluate the skill level of each team member and their contribution rate which is in direct correlation with the anticipated amount of Story Points the team member can complete in a given Iteration. Evaluations should be made on a regular basis and forecasts adjusted accordingly as it is reasonable to assume that the longer a team member is on the project the more likely it is that they will be able to increase their contribution rate. The team member's contribution rate is used to determine the impact to the Iteration(s) until the team

member is replaced. If the team member is replaced immediately then it is reasonable to assume the impact will be contained to a single Iteration, worst case two. If longer, then the ScrumMaster will need to use his or her judgment in the potential impact to the project. If the team member is not replaced, then a permanent impact should be represented. Figure 6 shows the impact on Velocity when a team member is removed in Iteration 3. As we can see the Velocity in Iteration 1 and 2 were 45 each. In Iteration 3 a team member was lost and Velocity was only 20 Points with a partial recovery in Iteration 4 with 40 points achieved.

The same calculation inverted shows the potential impact of team members being added. Since the calculation is inverted the contribution level of an additional team member will be minimal in the beginning but greater over time until they reach their full contribution level. Figure 7 shows a team member added during Iteration 3. The team's Velocity was again consistent at 45 for Iterations 1 and 2 but jumped by an additional 10 Points due to the new team member being added. This additional contribution appears in Iteration 4, but the data is not conclusive to determine if the new team member added additional value above the 10 Points as the team achieved the full scope of the project during Iteration 4.

When evaluating the addition of team members, the ScrumMaster needs to account for there being a certain threshold where the additional person's contribution rate diminishes as the team becomes larger. The ScrumMaster must rely on their experience and familiarity with the team dynamic to make this determination.

Additional factors can also cause an impact, positively or negatively, to team performance. Examples of specific factors are: scope changes in regards to functionality or features, time of project being reduced or increased or work to correct defects needing to be introduced immediately with no flexibility in scope being reduced for the current Iteration.

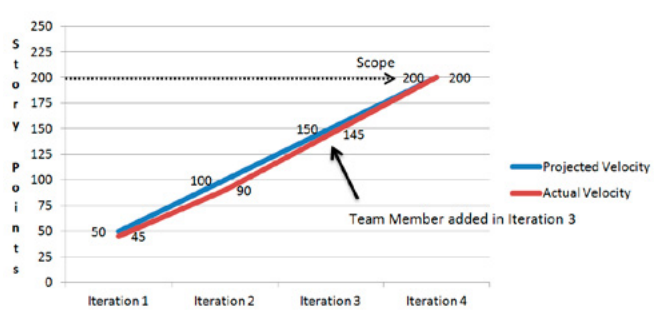


Figure 7. *Release Burnup with Addition of Team Member*

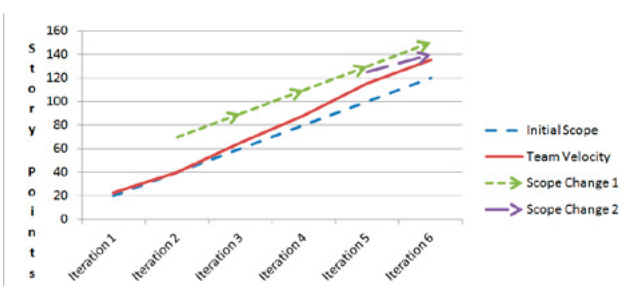


Figure 8. *Release Burnup with Scope Changes*

Other Uses: Tracking Scope Changes and Impact of Unaccounted for Factors

Many outside factors can positively or negatively impact a team. In reviewing one of the most prominent and familiar situations a Scrum team is likely to encounter, an increase in scope with no flexibility in future scope reduction or time added to the project. Team morale is negatively impacted since it negates the value of their estimates as previously reported to management. At times, overtime can resolve these situations but should be rare as burnout is likely to occur..

Figure 8 represents a view of several Iterations with two scope changes occurring. It is assumed that the team size and their capabilities were not impacted at any time and that the scope changes were introduced in time for the team to plan accordingly and did not interrupt an Iteration that was already underway. This Burnup can be interpreted in many ways. If no scope changes were introduced it appears that the team underestimated their capabilities and could have completed the project in Iteration 5. With the introduction of Scope Change 1 though we can see an impact to their performance when the change was introduced with a strong recovery afterwards but nowhere near where they could complete all the work before the final Iteration. With Scope Change 2 occurring and reducing scope the team is much closer to completing all the work in the final Iteration but would come up short. We can see that a slight increase in the team's velocity occurred when this reduction was implemented, most likely due to a morale boost due to the higher probability that the team would complete the project on time.

A ScrumMaster must ensure these types of occurrences are clearly documented in Burn Charts and that the information is re-iterated throughout the course of the project to keep management and others informed of the introduction of the unplanned event(s) and its long-term impact. This can lead to discussions of mitigation plans either in immediate or future Iterations in an effort to bring the project back on schedule. It is imperative that a ScrumMaster ensures this information is well documented in the case that justification or rationalization is needed in the future.

Other factors can significantly impact schedule and team morale, examples being:

- high priority bugs being found and needing immediate resolution,

- a competitor introducing a similar product thus driving the team to get a product to market quicker,
- organizational changes taking place and impacting team member's availability,
- difficulty in obtaining necessary tools to support the project (i.e. build environment hardware upgrade, software licenses, etc)
- PO not being readily accessible.

All of these factors and many more impact a project schedule and cause distractions that take away the team's focus on planned User Stories thus lowering their velocity for a period(s) of time. ScrumMasters should do their best to minimize these distractions if at all possible. Many will likely be outside their control so keeping team morale as positive as possible must be the ScrumMasters primary focus in order to continue the progress of the project.

Summary

When utilized correctly Burn Charts are effective to communicate different messages and status. ScrumMasters are responsible for creating and maintaining Burn Charts as well as interpreting the information for interested parties to ensure misinterpretations are mitigated if at all possible. ScrumMasters should work closely and regularly with the receivers of the information and tailor Burn Charts in such a fashion that communicates the most positive outlook possible to keep team morale high but at the same time not giving a false sense of the project's health. There is no correct cookie-cutter approach to effectively using Burn Charts, each team and organization's needs vary and it is the responsibility of the ScrumMaster to ensure the needs are met.

CHRIS MERRYMAN

Chris Merryman, MBA, PMP, is an avid Project Management practitioner and everything-Agile enthusiast. Chris has over 10 years of experience in the IT industry and has developed and worked on several complex software and hardware projects utilizing PMI's Project Management practices and Scrum methodology. Chris can be reached at www.linkedin.com/in/chrismerryman/.

Glossary

- Acceptance – the approval by the Product Owner that the User Story's features and/or functionality is complete and works as intended
- Burn Chart – a tool primarily used to track the status of Story Points across individual Iterations and the Release.
- Commitment – an event where the team agrees to deliver production-ready code for Acceptance no later than the end of the specified Iteration. The event takes place before the beginning of the Iteration in which the work is to be performed.
- Information Radiator – term introduced by Alistair Cockburn [4], a display where project information is posted for the team and other interested. Examples: stationary or mobile whiteboards.
- Iteration – the set time period in which the team works and delivers production-ready code and obtains Acceptance from the Product Owner. Routinely four weeks in duration. Iteration naming strategy is developed by team and tracked by name under the Release Burn Chart.
- Release – a product that is ready for market. A release is comprised of several Iterations.
- Scrum – an iterative-based software development methodology that incorporated Burndown charts early on as a tool to consistently report status and trends for different time intervals, also known as iterations.
- Story Points – estimates of the number of ideal days that a user story will take to complete coding, testing and be ready for Acceptance.
- Timeboxing – a method used to determine the fixed length of time of each Iteration.
- User Stories – features or functionality described in enough detail for the team to estimate and work to begin.
- Velocity – the sum of all Story Points that obtained Acceptance in previous Iteration. The team's velocity is used in burn charts when reviewing charts in relation to the status of a release or multiple iterations.

References

1. Calculating Sprint Burndown and Velocity of Work – *http://web.archive.org/web/20010503112119/www.controlchaos.com/sburndown.htm*
2. Definition of Burndown Chart – *http://guide.agilealliance.org/guide/burndown.html*
3. Measuring the Velocity of your Scrum Team – *http://www.versionone.com/Agile101/Agile-Scrum-Velocity/*
4. Ambler, Scott (12 April 2002). Agile Modeling: Effective Practices for eXtreme Programming and the Unified Process. John Wiley & Sons. pp. 12, 164, 363. ISBN 978-0-471-20282-0.

vixu.com

Vixu.com offers professionally supported website-management software that is written in Clojure and available under the Apache License. The developer of the software, Filip de Waard, can also help you bootstrap your own Clojure development project as an independent contractor.

Scaling to a Billion

Scaling your system to a support business growth requires architects and developers to adjust the way they look at their systems and processes.

In 2011, the late stage startup I was with sold and fulfilled eCommerce orders in an annual rate of half a billion dollars. After being purchased by a major brick-and-mortars retailer, our backend fulfillment systems were enhanced to fulfill not only the orders coming from our own site, but also to handle most of the fulfillment of orders placed on the retailers' site. With multiple sources of orders, we had to be ready for considerably more orders placed and processed. As the Principal Engineer of the group responsible for Order Processing and Payment systems, I led the technical design of our systems toward handling that elusive and quite challenging goal of enabling our systems to support one billion dollars in annual sales. This article is about some of the lessons that the team and I learned in the process.

Reducing Uncertainty under Pressure

When your application is directly responsible for revenue, you and your team will find yourself constantly under a microscope. When issues occur, escalations are distracting and time consuming; therefore, part of the effort in designing a system must go towards minimizing outliers and possible escalations. A stable, reliable system may be preferable to a more efficient system that runs in fits and breaks. When looking at performance and SLAs, it is not enough to minimize the average response or cycle time – minimizing the variance of those numbers is crucial.

Know Your Enemy

As the scale of your systems increase, the number and types of exceptional or unexpected behaviors increase. At the same time, your understanding of what the system is actually doing decreases.

As the system scales, there is a need for more complete and more actionable logging. Those two requirements are sometimes in conflict – the more verbose logs are, the harder they are to understand. To address this problem, invest in log filtering and analysis tools. One approach is to implement Structured Logging, where all logging statements have essentially the same format and are therefore machine-readable. A close cousin to logging is an effective monitoring system. Some events require human investigation and intervention, but involving the team can be distracting, demoralizing, and expensive. A good monitoring system requires a low false positive rate (do not alarm when nothing is wrong) and a very low false negative rate (do not miss alarms), but tuning alarm criteria to meet both of those goals is difficult. Every customer order is important, but it is impractical and demoralizing to wake up a developer at 2:00am to fix a single order, so monitoring systems must prioritize events based on the number of customers or requests impacted. As the system grows, manual handling of exceptional events becomes less reasonable. A machine failure is a likely event in any fairly-large cluster, so it should not require immediate human intervention. Failure of some less-trustworthy components should be treated similarly and the system should maintain SLAs without immediate intervention. As the cluster sizes grow, more classes of errors should fall into the 'automatic handling' category. In specific, a failure of one request (or one type of request) should be automatically isolated to not impact future requests, or at least not impact requests of a different type. If the system suffers from known issues (which may take weeks or months to address), the automatic handling system should be able to easily add appropriate mitigation. For example, if a new bug is introduced where purchase orders with more than 50 different products fail more often, but the bug is not immediately fixed – since those are rare, the team might want such orders (or such failing orders) to be automatically 'parked' and handled during normal business hours only, rather than trigger the existing alarms, since those failures are not an indication of a new problem.

Up Up and Away

When asked to scale a solution, the first concepts that leaps to the minds of developers is momentary load. Measured in "requests per second', scaling consists in handling more requests in the same time. But how many more? And when? It is likely that the load on your system varies through the day, between weekdays and weekends, and between rush times and ordinary days. For drugstore, like many other eCommerce retailers, Black Friday (and Cyber Monday) represented an annual peak. Yours may differ – but you should know when your requests hit max load, and what load you expect to handle. Back-end requests can often be queued,

but queuing introduces delays – your service development must be guided by your SLA (Service Level Agreement): is it OK to delay processing some requests for 15 seconds on peak times? How about 15 minutes?

A few tricks to help optimize systems quickly are:

- Minimize data flow. Moving data around in memory, or from disk takes time. Check the columns fetched in your SQL query and slim them down. Verify that your database table only contains required rows, and that unnecessary (or old) rows are purged or archived. Considering covering indexes for common queries and reduce the amount of data (such as unnecessary fields) passed in web service calls.
- Scaling horizontally means adding more machines. It is easier to design a system for horizontal scaling if your services are stateless. Soft state (cached data) is usually OK, but consider a shared or distributed cache if the same data ends up cached on multiple machines.
- Seek out and eliminate all single point of failures. The same search will likely identify some of your choke points – the servers that have to handle -every- requests. Consider alternatives.
- Scale horizontally or partition your data. Either plan for many machines to process your workload at once, where each machine has access to the entire data, or divide your data into partitions, and have a separate set of machines process each. Either approach has pros and cons – understand your tradeoffs.
- Simplify your solutions. Complex solutions are hard to maintain or even get right and they are harder to optimize.

Nothing is Free

Understanding architectural tradeoffs is important. For example, consider queuing solutions and Enterprise Message Bus to enhance reliability or enable batching, but use direct Web Service calls when turnaround time is meaningful.

Exceptional Service

With millions of transactions, each involving multiple touch points, the unexpected is the norm. Subtle bugs may impact only a small percentage of request and skip testing; requirement analysis or implementation may miss some input combinations; and timing issues or hardware failures may cause unexpected states to manifest. All these unexpected issues require good, tiered. exception handling.

First, code must be constructed with error handling in mind. Exceptions need to be caught and handled, issues should be automatically isolated to the minimal set of related requests, and nothing should bring down the system or stop the processing train.

But, even with the best coding practices in time, issues will creep up at the most inconvenient times. All large scale systems require a tiered level of off-hours support (on-call), and the participation of trained engineers and programmers in the process. Alarm levels should be set appropriately to balance reducing system risk and support personnel workload. However, handling one-off errors and investigating suspected issues can waste precious time, cause employee dissatisfaction and work-life balance issues, and increase the churn in the team. Best people practices include investing in continuously improving the system, automatically delaying handling of issues impacting only a few transactions to business hours (e.g., by setting high threshold for alarms), and allowing employees to receive comp time if on-call issues resulted in off business-hours work.

Final Words

Optimizing engineering metrics is never sufficient. To support a high transaction value system, the architects must understand and optimize the metrics the business cares about. In some businesses, low error rate is crucial. In others, low turnaround time. Some businesses care about average response time while others cannot tolerate slower response time even for a smaller fraction of the requests. The metrics measured – average latency or P90 (a measure of the experience of the worst 10% of requests) latencies, request error rate or down time, must fit the business the company is in.

Also, above all, understand your assumptions and state them clearly. Handling 'five times the order volume' may sound specific – but does your tests assume that each order has less than 10 items? Or, that orders arrive in a constant rate over an hour rather than in bursts due to batching in other systems? Or, that other systems do not cause load or lock contention on the database? Misunderstanding your requirements or assumptions may result in perfectly engineered systems that do not help the business grow as planned.

Above all, remember that scaling a system is hard. Systems do not scale linearly, and in many cases, handling twice the load requires more than twice the computing resources, developer time, and stabilization period. Advantages to scale exist – but they take time to materialize.

YANIV PESSACH

Yaniv Pessach is a software architect living in Bellevue, WA. Throughout the years, he has worked for multiple S&P 500 companies, as well as several startups. Yaniv received his graduate degree from Harvard University, where his research focused on distributed systems. You can find more about Yaniv on his website, www.yanivpessach.com, or contact him through his Linkedin page at http://www.linkedin.com/in/yanivpessach.

The 6 Critical Things

to Consider When Partnering with a Mobile Developer

The explosion of mobile technology has prompted businesses of all sizes to embrace mobile applications in hopes of gaining a competitive edge in their industries. However, in the rush to move forward, businesses often fail to vet mobile developers adequately. Partnerships formed without due diligence end up moving businesses away from the acclaimed mobile benefits rather than towards them, ultimately costing them time and money.

The majority of businesses look at a mobile development project as a destination, notes Sunil Jagani, Founder and Chief Technology Officer of AllianceTek, an information technology company based in Malvern, Pennsylvania. "But most software projects are a journey that must be smartly managed."

> *"The majority of businesses look at a mobile development project as a destination. The fact of the matter is that most software projects are a journey that must be smartly managed."*
>
> *Sunil Jagani*

Jagani should know. After earning his Masters in Software Engineering from Pennsylvania State University, Jagani jumped into the computer software industry with both feet. He spent six years working as a Senior Technical Lead at Advanta before leaving in 2006 to start his own company. His perspective – both from behind the monitor and the president's desk – has taught him the value of identifying success as something that happens along the way. "Every phase of the project must consist of start-to-finish milestones," Jagani asserts. "The success of one milestone will make way for the success of the next, and so on until the project's completion."

How, then, to choose a mobile technology partner from a crowd of prospective partners all clamoring for your business? In addition to gathering information on the cornerstones of a reliable partnership (experience, references, and expertise), there are six critical mobile-technology-specific considerations businesses should examine.

1 Does your mobile partner have multi-platform experience?

In the past, when developing web or desktop applications, one or two robust application versions would be enough to accommodate users of most browsers and operating systems. However, there are many form factors in the mobile marketplace, including the iPhone, Android, and BlackBerry. Tablet devices such as the iPad also require a high level of expertise and know-how. When looking for a mobile developer, find one who has experience in accommodating multiple form factors. Failing to do so could alienate a significant portion of your potential users.

2 Can your mobile partner provide back-end system integration?

Many mobile developers are versed in creating isolated applications packaged for general use. But it is critical that the mobile application's architecture connect to your organization's back-end systems. That's because there's a distinct gap between a generic application and your existing systems, forcing parts of the application to be recreated in order to use them effectively. A mobile developer needs to be able to assess your existing architecture and ensure that the mobile application is designed to integrate seamlessly with your back-end systems.

3 Is your mobile partner capable of contributing new ideas to your mobile strategy?

Mobilizing is more than simply taking your existing content and features and making them accessible on mobile devices. Because at its foundation, this new platform is about more than mobility. It's nothing short of a game changer that is radically redefining the way users interface and interact with applications. Businesses that fail to design for the unique capabilities of the platform do so at their own risk.

When choosing a mobile developer, go with one that is able to show you how mobile technology can be used to develop unique applications that use the platforms' features, such as touch interfaces, location awareness, and application connectivity.

4 Is your mobile partner a business solutions architect?

A mobile development partner should bring additional value and insight to a project. If a mobile developer simply builds according to your instructions, they're fulfilling the role of a service provider, not a solutions architect. „A mobile technology partner should be a partner in the truest sense of the word," says Jagani. „They should add their own ideas to achieve your objectives that include insight from a mobile technology expert's perspective."

Even if you possess a basic application concept, the business solutions architect should help you to expand on the concept and perfect it.

5 Does your mobile partner offer value identification and prioritization?

Let's face it. At the end of the day, every business has a mandate to ensure a positive return on their investment in mobile technology. A reliable mobile development partner plays a critical role in achieving return on investment by identifying the lowest hanging fruit, and discouraging extra features that won't yield benefits worthy of the time and effort it takes to create them. According to Jagani, „It takes the understanding of both a business solutions architect and a software architect to develop an effective mobile solution that meets business objectives." Sometimes this means breaking development requirements into smaller parts and prioritizing them based on which can be developed efficiently and effectively first.

> *"If a mobile developer simply builds what you tell them, they're fulfilling the role of a service provider, not a solutions partner."*

6 Does your mobile partner assume end-to-end responsibility for the project?

A mobile project can only be successful if there is a clear understanding of the business strategy, application goals, requirements, and component and feature priorities.

From inception to deployment, the mobile partner should take responsibility for all considerations and have a clear view of the big picture objectives -- including those that may only come into play down the road. "It's the mobile-development partner's responsibility to identify the core features needed to get a working first version and what features should be rolled out in future versions," Jagani observed. "Placing this responsibility squarely on your mobile partner's shoulders ensures that he has skin in the game, and your mutual success is priority number one."

Central to each of these six points is that a mobile partner should be, above all, a business solution architect. This means they can ascertain your business' unique challenges and objectives and apply their expertise to develop a mobile solution tailored to your needs. How to ensure you find a partner who will do this? "A mobile partner should align their success with your own so they are incentivized to help you achieve the best results for the lowest costs," Jagani concludes.

You must demand that your mobile partner put their money where their mouth is. No matter how rapidly technology changes the way we do business, fundamentals such as this remain constant.

Database monitoring that requires less effort than any other solution.

No configuration. No hardware to manage. Deploys in minutes.

- » MongoDB
- » MySQL
- » Oracle
- » PostgreSQL
- » DB2
- » NoSQL

..and more!

Instant monitoring with graphing
Everything critical on your database monitored and graphed according to best practices.

SaaS Architecture
Month-to-month billing. No upfront costs. No hardware to buy. No systems to maintain.

Real-time Alerting
Get alerts via email, SMS, or phone call that tell you what's wrong before damage has been done.

Reliability
LogicMonitor's hosted monitoring won't leave you blind when your network goes down.

Performance Tuning
Interactive graphs with up to 1 year of trending data will show you exactly what's going on.

Monitor Your Entire Infrastructure
Don't stop at the database. Monitor your entire infrastructure with one tool.

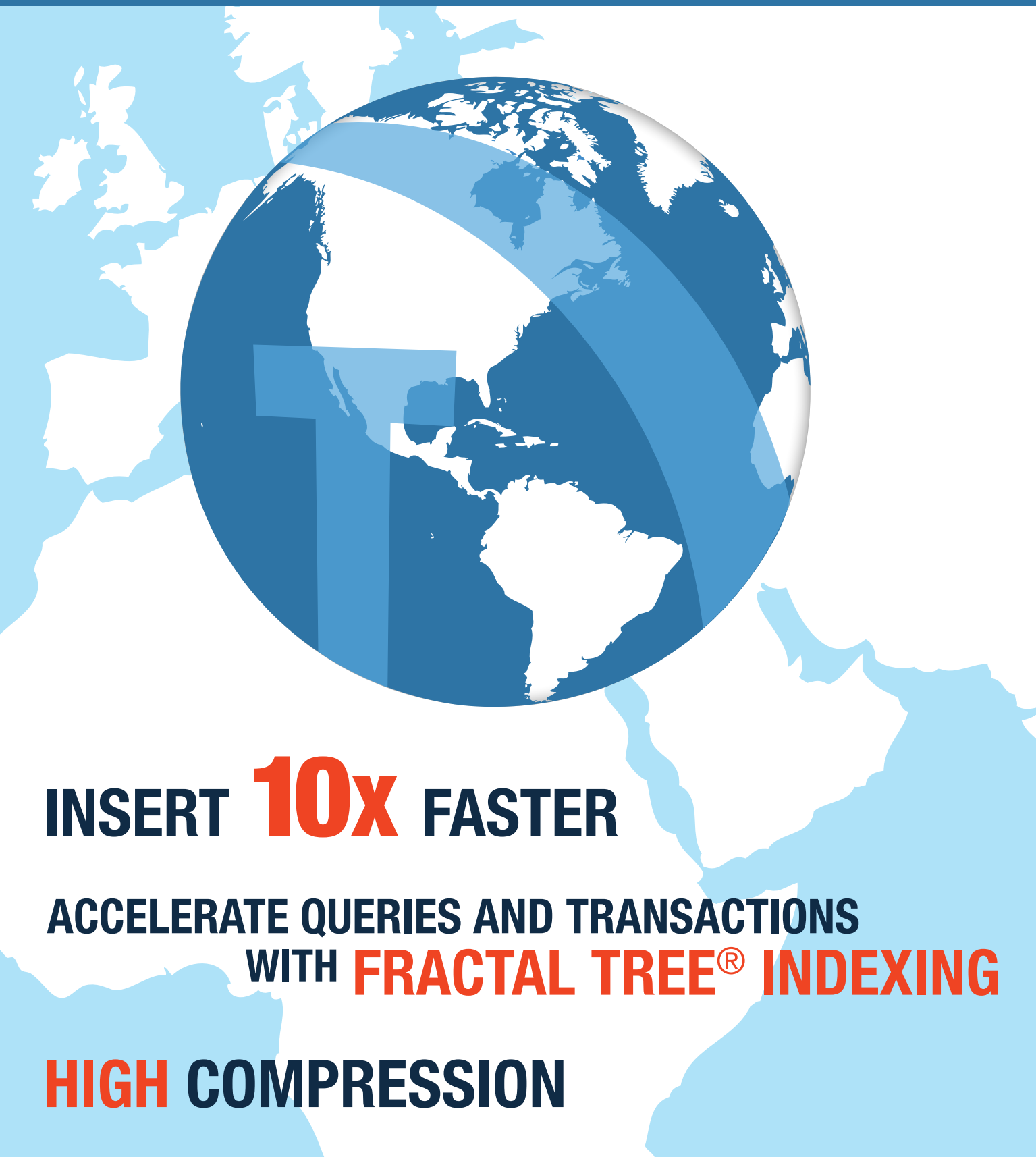
Learn more at www.LogicMonitor.com/sdj

LogicMonitor

TALK TO US: 888-415-6442

Tokutek®

DELIVERING DATABASE PERFORMANCE AT SCALE



INSERT 10X FASTER

ACCELERATE QUERIES AND TRANSACTIONS WITH FRACTAL TREE® INDEXING

HIGH COMPRESSION

http://tokutek.com