

The NeceMoon Album

Technologies and Strategies to Keep Moving Forward



Volume 2 Full Moon (Hardcore)

By
Necemon Yai

Featuring
Darren Mart
Jean-Patrick Ehouman
and Holty Sow

TheAlbum.NeceMoon.com
First Edition | Evasium ®



The NeceMoon Album

Technologies and Strategies to Keep Moving Forward

By Necemon Yai

First edition

Published by Evasium ®

April 2018

London, UK

TheAlbum.NeceMoon.com

The contents of this file are protected under the **UK Copyright, Designs and Patents Act 1988**.

License

This file is free to distribute and give away to as many people as you would like.

I only ask that you do not sell it or publish the content onto any website.

If you use any quotes from this document, give me credit and link to the original file.

If you write a book and I quote you, I will give you credit and links too.

© Necemon Yai

necemon@gmail.com

www.necemonyai.com

All Rights Reserved.

Version 1.0.7.235

To the ones I lost, to the ones I got back

For each one that gets lost, ten get laid back

Table of Contents

Introduction.....	6
For All Practical Purposes	9
Volume 2: Full Moon (Hardcore)	10
Chapter 6: Software Development and Engineering	11
8 reasons why you would enjoy being a programmer	12
Writing and Programming: Pretty Much The Same Thing (by Jean-Patrick Ehouman)	14
On Technical Orientation : 5 Basic Considerations When Starting To Code	15
4 Substantial Tactics Guaranteed To Boost Your Game Creation Skills	17
10 Years of Programming	19
Should you build it if nobody comes? (by Darren Mart)	23
Useful Job Search Websites For Programmers In The UK	27
Microsoft Professional Certifications for Developers	30
Chapter 7: C# .NET Programming	32
Why I like C# so much.....	33
C# Free Learning Resources For Beginners And Professionals.....	36
How to Track Mysterious Bugs with Visual Studio	38
How to Debug a Windows Service in Visual Studio (by Holty Sow)	40
My Essentials : Top 12 Tech Courses on Pluralsight for .NET Developers.....	43
Deploying a Web App to IIS / Windows Server In 7 Clever Steps	47
Launching Your Application At Windows Startup Without Hacking The Registry (by Holty Sow)	49
My 12 Favorite Entity Framework Tricks.....	51
ASP.NET MVC: Avoid Polluting The Model Of The View With Error Messages (by Holty Sow)	55
Limiting The Execution Of An Action To AJAX Requests Only (by Holty Sow)	57
[Interview] In The Bubble Of Holty Sow: "for more efficiency, I prefer to work in an agile team"	59
The Xamarin Revolution	61
Top 8 Tricks with Xamarin Programming	63
Chapter 8: Epic Prototypes, Classic Projects, Historic Genre	65
Inside Bavardica - Part 1 : Discussion of the subject areas	66
Inside Bavardica - Part 2 : Progress and Achievement of the Project	69
Inside Bavardica - Part 3 : Design of the 2D characters.....	75
Inside Bavardica - Part 4 : Scope for future improvements	79
10 things I learned from building Bavardica.....	81
Babi Fraya: The Hot Winter Illusion.....	83

Relaunch : Kpakpatoya 2.0 as an Interactive Platform.....	86
Flux Project : Social News Web Applications.....	92
Stay up to date with your favorite topic on social platforms.....	94
Furtivue (Or How To Send Furtive Messages Like A Ninja)	96
Chapter 9: Research and Case Studies.....	97
Phidgets: First Steps In Robotics?	98
Paper Prototyping.....	105
Dual Shock 3 - Part 1: History of game controllers.....	107
Dual Shock 3 - Part 2 : The amazing story of the Dual Shock	111
Dual Shock 3 - Part 3: Device Analysis and Specification	113
Business at the speed of game production: OpenGL or XNA?	116
GOOSE (GOOgle Supply search Engine)	117
Final Reminder.....	121
Conclusion	122
Share The Album with Your Mates.....	123

Introduction

About The NeceMoon Album: what is this all about?

[The NeceMoon™](#) is a Blog about Technology and Strategy. [The Album](#) is The Best-Of, a compilation of the most popular articles on The NeceMoon™.

The main objective of The NeceMoon™ is to share tips and insights on a sensible range of topics, in order to let others learn from my mistakes and successes, and hopefully to make things easier for the next person. The main objective of The Album is to promote an optimal access to that content. The Blog format does not always do justice to techno-strategic content. Originally, Blogs were designed in a journalistic spirit and are more suitable to chronological events and (more or less trivial) discussions around daily news. Even if the usefulness and the importance of an analysis persist in time, it becomes almost impossible to find and difficult to consult, as more articles keep stacking.

That is why the best articles have been cherry-picked, reviewed and arranged in a logical order that better matches the layout of a book. The Album is free of charge.

The NeceMoon™ can be accessed from NeceMoon.com (or necemonyai.com/blog)

The NeceMoon™ Album can be downloaded in various file formats, in full from TheAlbum.NeceMoon.com (or necemonyai.com/Blog/page/The-Album.aspx). The available formats are PDF, EPUB, MOBI/AZW3/KF8 (Amazon Kindle) and MP3. Furthermore, the various chapters and volumes can be downloaded independently/separately, according to your interests.

About the Author: who is Necemon Yai?



I am a Software Development Engineer extensively involved in Microsoft .NET technologies. Full time developer. Part time digital artist, strategist, essayist and entrepreneur. I majored in computer science from NIIT, Christ University and Swansea University (Master of Engineering, Computing).

At the time of publishing this, I have worked for a Europe leading E-Commerce company, a major UK financial group, the General Electric global corporation and a few tech start-ups that you probably never heard of.

Over the past decade or so, I have been running The NeceMoon blog, where I describe my experiences, my observations and my reflexions. I mostly talk about Technology and Strategy. Here I share my most popular articles.

About the Contributors: who is in your War Council?

I invited the best writers in my network to include some contributions in this book, especially some of their most relevant insights in terms of Technology and Strategy. These top authors are, Ahou The African Chick, Antoine Mian, Cyriac Gbogou, Darren Mart, Edith Brou, Holty Sow, Israel Yoroba, Jean Luc Houedanou, Jean-Patrick Ehouman, Karen Kakou, Monty Oum, Nanda Seye, Nnenna Nwakanma, Olivier Madiba, Vanessa Lecosson and Yehni Djidji.

Along with their respective writings, you can find links to their own web pages. In addition, most of these contributors introduce themselves and provide you with some tactics in our exclusive interviews that you will also find in this book.

About You, Dear Reader: who is this book for? What's in for you?

In The Album, there are 9 chapters organised in 2 volumes. Each chapter deals with a specific topic. You don't have to read everything. If you are interested (to one extend or another) in one or more of these topics, you would possibly appreciate the related chapter(s):

Volume 1: Moon Light (softcore)

- Chapter 1: Strategy and Tactics
- Chapter 2: Digital Marketing and Web Visualisation
- Chapter 3: Corporate Worlds and Emerging Markets
- Chapter 4: Quick Wins, Tricks and Tips
- Chapter 5: Transition - Extra Thoughts and Sharp Fantasy

Volume 2: Full Moon (hardcore)

- Chapter 6: Software Development and Engineering
- Chapter 7: C# .NET Programming
- Chapter 8: Epic Prototypes, Classic Projects, Historic Genre
- Chapter 9: Research and Case Studies

If you want to, you can download and read only the chapter(s) and volume(s) that you are interested in. Several file formats are available on TheAlbum.NeceMoon.com (or necemonyai.com/blog/page/The-Album.aspx)

All the Web links in this document are working, feel very welcome to click on them.



For All Practical Purposes

This document contains the Volume 2 of The Album. It includes 4 chapters. If you care, 5 additional chapters are available in the Volume 1. Depending on your interests, you may download, (re-)read or share any of the various chapters and volumes independently/separately. The PDF, EPUB, MOBI/AZW3/KF8 (Amazon Kindle) and MP3 formats are available.

To get them, just click on any of the links you like below or go to TheAlbum.NeceMoon.com (or necemonyai.com/Blog/page/The-Album.aspx)

The NeceMoon Album (complete)

Volume 1: Moon Light (softcore)

[Chapter 1: Strategy and Tactics](#)

[Chapter 2: Digital Marketing and Web Visualisation](#)

[Chapter 3: Corporate Worlds and Emerging Markets](#)

[Chapter 4: Quick Wins, Tricks and Tips](#)

[Chapter 5: Transition - Extra Thoughts and Sharp Fantasy](#)

Volume 2: Full Moon (hardcore)

[Chapter 6: Software Development and Engineering](#)

[Chapter 7: C# .NET Programming](#)

[Chapter 8: Epic Prototypes, Classic Projects, Historic Genre](#)

[Chapter 9: Research and Case Studies](#)

The Album is available in French as well at Album.NeceMoon.com (or necemonyai.com/Blog/page/L-Album.aspx)

Volume 2

Full Moon

(Hardcore)

In this volume:

Chapter 6: Software Development and Engineering

Chapter 7: C# .NET Programming

Chapter 8: Epic Prototypes, Classic Projects, Historic Genre

Chapter 9: Research and Case Studies

Featuring Darren Mart, Holty Sow and Jean-Patrick Ehouman

Chapter 6

Software Development and Engineering

Featuring Darren Mart and Jean-Patrick Ehouman



8 reasons why you would enjoy being a programmer

By [Necemon](#)



If you are a programmer, there are probably a lot of reasons that motivate you to do what you do. I hope you find here some more motivations.

If you [want to be a programmer](#), you may find here some more reasons to go for that awesome path.

However, I guess I am writing this mainly for [those who don't know what to do with their life](#). Here are some clues about a job that is fun, useful and satisfying to many extents.

From the top of my mind, here are 8 reasons why you would enjoy being a programmer. I hope this inspire you.

1. Programming makes your dreams come true. When you understand programming, you can [give life to your thoughts](#) by applying them to real life. You can literally create things.
2. Programming is the [ultimate form of interactive art](#). You can make software, websites and games that others can play with. So you can talk to them indirectly and they can talk back. No other art form is quite this interactive. While drawing, painting, movies and music go to the audience (in one direction), code goes both ways.
3. That's the kind of job you can do from anywhere. From your couch, from home, from office, on travel, whichever country... The only things you need are a computer and your brain.
4. It's easy to learn. There are [tons of ressources available online, many of them are free](#). Also, there are many online communities that can support you through forums, chats, emails, etc.

5. You don't have to rely on anyone to do programming. You may have a chance to do it [for a company or for a research programme](#). Even if it's not in a corporate job, you can work in a team. And even if you don't find a team that fits you, [you can work by yourself](#). Also, you don't need much money to get started.

6. You can't have enough of it. You can't get bored. [Requirements and technology](#) are moving up so fast that you always face new challenges.

7. It's a field of meritocracy. They know you don't fake your skills. You [know what you know](#). You do what you can do and you get a fair recognition for it.

8. It's fun !

N.

Writing and Programming: Pretty Much The Same Thing

By [Jean-Patrick Ehouman](#)

5 years ago, if I was told that I could [run a blog](#), I would not have believed it. As a [software engineer](#), I was spending more time [designing and writing programs](#). However, a priori, nothing suggested that I [could write good articles](#) too.

In hindsight, I can say that these two activities have more similarities than differentiation points. When you write, you create, you fill a blank, you give life. Similarly, when you design a program, you create a system that will be used on a daily basis. So in both cases you need to deeply understand the reader or the user of your creation.

Putting yourself in their shoes leads you to be imaginative and creative like a painter or a pianist. So, when I am asked about the [fundamentals required](#) to learn how to make good computer programs or software, I ask the interested party if he has ever written a letter, a short story, an article, etc. If the answer is "yes", then the person already has prerequisites to learn how to write programs. If not, they can still try to write [a short story](#) and assess their abilities to create or innovate.

[Coding is just writing.](#)

On Technical Orientation : 5 Basic Considerations When Starting To Code

By [Necemon](#)



A [fresher](#) sent me the following query:

Hi Necemon, thank you for accepting my invite. I am new to the IT field and I would appreciate if you could give me some guidance. Based on your work experience, can you tell me what companies look for in a computer guy ?

I heard about you from senior students at [Christ University](#), I am in Bangalore and I like computer science but I do not know what to learn and how to begin.

Actually, I think [I like programming](#) but I am told that the C language is no longer relevant, and [I am also told about Ruby, C#, Python, etc.](#) I'm confused.

Please don't get confused. Technology is simply a way to solve a problem or to achieve a goal. What is your goal?

Create apps? What apps do you want to create and why?

It's a bit as if you come to me to ask me what vehicle you should be driving. If I ask you what you want to do with that vehicle, you wouldn't just tell me that you just want to move away, right? I know you want to move... My question is, where are you going?

What companies are looking for ? Ok, I fully understand what you are asking here. You want to make sure your education will guaranty [an interesting job](#) later in the [IT development Industry](#). Obviously, I could tell you that a certain technology T is in high demand right now, but it's not that simple. There are a few other things to consider:

1. The requirements may vary with location (country or region). The hottest jobs in the US are not necessarily that popular in India. So unless you know already where you are going to work, it's not that easy to target on a trend basis.
2. The demand changes with time. What is relevant today may not be (as) prevalent tomorrow. The technologies evolve and replace each other. So what's fashionable now might be different from what will be popular by the time you get your degree.

3. You might not like the technology in vogue or the uses of that technology. If I tell you that a given technology T is in high demand, that it allows you to locate and fix bugs / errors in a super boring / huge / complicated banking system, plus there are plenty of calculations... What if you don't like calculations? Are you still going to embrace this technology and to accept this path for the rest of your career?

4. Even if we consider only one city and a given time, various companies are seeking different things, depending on what they do. There is no perfect technology that is better than all others in all areas. Each technology has its own sets of advantages and disadvantages. C and C++ may be better than Ruby at a few things (and vice versa), Python is better than C# in some respect (and vice versa), etc.

5. As I said above, the technology is just a mean to get somewhere. When you consider [Facebook](#) for example, most users don't care if it was built with [PHP, C, Java, Perl, C# or Python](#). What is important for people is, how the site or application can help them in their lives.

I think that's where you should start. What strengths and assets do you already have? (don't tell me you don't have any). What contribution do you intend to come up with for your family, your friends, your community, your country, and for the world? And what do you expect in return?

If you do not know what to do with your life, some time ago I wrote an article that might inspire you : [What will you do in life?](#) Take time to reflect on your ambitions and we can talk about the resources you will need.

If you know WHAT you want to do, it would be easier for me to tell you HOW to do it.

Let's speak soon,

N.

UPDATE - [Shabbir Kahodawala](#) shared a few clever insights on this matter :

I agree very much with your response to the fresher - about the need to realise and work on his talents and concentrate on writing good code and immersive UI.

I would like to add a few points as well taking the perspective that every Indian student goes through the same dilemma due to lack of job oriented education, because institutions focus on technical oriented education.

IT is not only about writing code. Think of it as a factory where there is Marketing, Client Requirement gathering, Planning, Product development, Product Testing, Infrastructure planning, Product Deployment, Product Maintenance, Customer Support, Issue Resolution and Product Improvements.

Each of these creates many job opportunities for IT students and one needs to understand where his strengths lie. He can do that by talking to IT professionals who can introduce to other professionals in each department and willing to share what skills they require on today's world. That will make his goals more clear.

Secondly, a competitive IT professional should always have his basics right. To be able to write good and neat code. He should be always able to visualize a requirement into an algorithm and then the language syntax that he uses can always change. That's why colleges teach C & Java (object oriented) because these help a coder develop his basics about Data Handling, Functions, Objects, Classes and runtime environment.

4 Substantial Tactics Guaranteed To Boost Your Game Creation Skills

By [Necemon](#)

A user of [one of my web applications](#) sent in the following question:

Hi my name is T. and I am 15, when I am older I want to be a part of the gaming industry and I am just wondering can you guys help with that, can you give me any tips or any experience?

Would posting to your website help me out in this? I would love to hear your reply, it would really help me out.



Here was my reply:

There is a lot to say on the topic but for today I will try to keep it simple and give you some practical advice you can start using right away.

1. Read a lot on the topics you are interested in. Read every day if you can. There are tons of free resources available online, some of my favorite sites and blogs for game creation are:

[gdne.ws](#)

[lostgarden.com](#)

[procworld.blogspot.co.uk](#)

[whatgamesare.com](#)

[gamestudies.org](#)

[designer-notes.com](#)

[higherorderfun.com](#)

[gamecareerguide.com](#)

[webwargaming.org](#)

[raphkoster.com/gaming](#)

[gamedevelopment.tutsplus.com](#)

[nwn.blogs.com](#)

2. **Establish specific goals.** You may have noticed that in point 1, I only gave you a small portion of what's out there, but it's quite an extensive read already. [We are living exponential times](#), you can't do and learn everything by yourself at once. You have to be specific about what you want. Vague plans provoke vague results. What is it that you want to do in the video game industry? There are so many options, you may want to do some research on your options.

3. **Pick your stars.** To bounce back from the previous point, after you make a list of the skills you want to master, do some research on people who are already really, really good at that. Find some role models, see how they started and how they made it. Combine their tricks and throw in your own style to develop your own skills. It may take months but if you practice consistently, you will get really good too. Some of the people you can check out:

Game development:

- [Shigeru Miyamoto](#)
- [Hideo Kojima](#)
- [Notch \(Markus Persson\)](#)

Digital artists:

- [Akira Toriyama](#)
- [Masashi Kishimoto](#)
- [Monty Oum](#)

Scenario writers:

- [J.K. Rowling](#)
- [Stephen King](#)
- [Tom Clancy](#)

4. **Make a game.** No, [you are not too young](#) to start. It doesn't have to be anything big and right now, you can at least start learning. The earlier you start, the more time you will have to practice and the sooner you will become excellent. You can start with something very basic or at least start reading about it. If you want advice on how to start, how to find tutorials and resources, I can help.

On a related note, I made a game myself : [babifraya.com](#) It's not great and I am working on an improved version. But the important thing is [to start and keep practicing](#) :-)

In short, educate yourself consistently.

As to whether posting to [degammage.com](#) would help, my honest answer is: maybe.

You see, it's a new project with a small community and it's too early to say how successful it will be.

However, there will be some fresh content everyday, so there will be things for you to learn, and you may also visit [flux.evasium.com](#) which is even more into virtual worlds, education and technology.

Your [Evasium](#) account will work on there too.

Additionally, you may want to join other game forums and communities, like the ones I listed above. The more you learn, the better!

10 Years of Programming

By [Necemon](#)



I [started programming](#) when I was in high school. I was taught Pascal and HTML among other things. But I was also taking some summer programming classes in an IT institute. That's where I learned Visual Basic and software/database design (during those days, we were using MERISE (Méthode d'Etude et de Réalisation Informatique pour les Systèmes d'Entreprise)). I eventually got Visual Basic installed on my home computer and a book from which I could practice alone, with books.

But, it's only after I graduated from high school, that I started [learning and applying C#](#) almost full time during higher studies, and that's what I considered to be my real start into programming and software engineering.

I learned a few lessons over the past decade, and I thought I would take a moment to gather my thoughts on these things. It took me about ten years and a lot of experimentation to figure out some of this.

1. Learning a programming language is the easy part: be aware and beware of the platforms

Take C# for example. Learning the C# language is not difficult. If you already have a good understanding of computer language fundamentals, and if you have some experience in other object-oriented languages, [you can become a competent C# programmer within a few days](#), at least as far as the language itself is concerned. However, the real price to pay is not about the language, it's about the platform. To develop with C# on .NET, you need to know:

- the .NET framework
- [one or more .NET technologies](#) such as ASP.NET or WPF
- and the Visual Studio development environment.

The time required to become proficient in .NET development is usually measured in months, even for an experienced developer. [Learning a platform](#) is always more expensive than learning a specific language, therefore choosing the platform is the most crucial decision.

Learning always has a cost and this cost is one of the key factors to consider when choosing the technology you want to learn. The real cost of learning is in time, learning always takes time. Since you do not have time to learn everything, it is important to think strategically about what you want to learn. And since the languages are easy, it's the platforms that you have to be careful about: the technologies associated with the language, the development and deployment tools, the operating systems, and other infrastructures.

2. I repeat, learning a programming language is the easy part: meet the underlying concepts of software engineering

The syntax itself, the words you use when applying the language are relatively simple and you can easily pick them up as you go. However, that's far from being enough [for producing quality code](#), which often involve OOP and SOLID principles, unit testing, design patterns, TDD, BDD, and other technical concepts which are beyond the scope of this article. Anyway...

3. Actually writing code is only one (small) part of the job

A software engineer is often expected to be involved into technology research, tools and projects configurations, DevOps and admin tasks, debugging and testing procedures, documentation, and technical debt (fixing and refactoring existing code). Also, they have to be thinking about solutions and designing systems : [sometimes the most important work is done away from a keyboard](#).

4. Tried and true : old and boring is sometimes the best.

It's not so much old vs new, or cool vs boring, but rather the thing you are most experienced with. As they say, I trust not the developer who practiced 1000 technologies once, but I trust the developer who practiced the relevant technology 1000 times. If the goal is to "just build the damn thing", go with a stack you would be most productive in.

For example, one of my contacts is making \$25,000 per month with a SaaS that was built on boring ASP.NET+SQL Server+Angular 1 because that's what he knew. he hosts it on Windows because he knows how to make it fast and secure. He succeeded by focusing all his time on building the features that clients were asking for, instead of learning fancy tech.

It's important to realise that [the technological treadmill never stops](#). Yet another JavaScript framework could have been launched while you are reading this. Today's cutting edge tech didn't even exist when I was getting started ([EF Code First](#), [Xamarin](#), ASP.NET Core, Razor), and this leads us to the 2 next points.

5. Focus on sustainable technologies

The only constant in the world is change. Actions and time management is an important skill for developers, particularly because we are on a technological treadmill that keeps moving or even accelerating.

For example, Web technologies that were popular around the year 2000 (Flash, ASP Classic and Java Applets) are becoming almost obsolete and decreasingly marketable. Today, we are talking about ASP.NET Core, SignalR, Angular2, React and VueJS. None of its technologies existed in the year 2000, and these new technologies are likely to be obsolete within 10 years.

What hasn't really changed? The fundamentals of languages such as C++/C#, their implementations of algorithms and their principles are still relevant after several decades. [If you master the basics of a stable system, you could adapt to change better](#), you could appreciate it and use it to evolve.

6. Balance between exploration and exploitation

Exploration is about [learning new things, studying new techniques, reading books, watching video tutorials, practicing and improving skills](#). Exploitation however, is to take advantage of what we already know [to solve real life issues](#). It's about thinking creatively about ways to use the knowledge we already have to create value for others.

So yes, both of these tasks are both necessary and important. The risk is to be too focused on either activity.

Too much exploration, and you will never achieve a useful level of expertise in the chosen technology.

There is a huge opportunity cost with this kind of light learning, because, although [it expands your mind](#), the time it takes implies that you don't really improve on the skills that you have already acquired.

On the other hand, too much exploitation can keep you from evolving in new technologies, and can limit your employment opportunities.

7. It's easy to be great... It's hard to be consistent

It's easy to be great for 2 minutes. It's hard to stay great constantly, every day.

When you have a good idea for a new project, you feel a great desire to start researching, designing and programming. You feel a rush to turn your idea into something real and you become super productive. But the problem is that this motivation fades over time.

Yes, it's fun and it's easy to have new ideas and start working on them. But then there are the efforts to be made, the adjustments, the launch, the maintenance, the corrections, the improvements, and so on. Over several months. This is where it gets hard. It is hard to stay focused on the same idea, on the same project for months and years. [It takes a lot of discipline](#).

It's easy to be great. It's hard to be consistent.

8. Diversify your skills

Don't be just a programmer, become an Expert Who Programs, an expert in [another relevant field that you are passionate about](#). You can be an entrepreneur, a project manager, a Big Data scientist, a researcher, a security specialist, etc. If you are an Expert Who Program, in addition to being able to program (maybe full-time), you also have [an additional credibility](#) that is related to [something other than software engineering](#). Hence the importance of getting an education. If you go to university and you already know how to program, you probably won't learn much about programming. That does not mean you should not go to these schools. You will need some culture, and universities are great places to get that. You acquire culture by [studying and understanding the world](#) that humans have created, from different angles. It would be difficult to acquire this kind of knowledge if you do nothing but study programming.

9. Pick your niches and standout

The smaller the niche you choose, the greater your chance of being viewed as a standout in your field. It's very hard for a developer to become a standout in "PHP Web development". They're great, versatile, useful, but not noteworthy. One developer who knows how to work with these technologies, feels they are easily replaceable because there are so many out there with a comparable skill set. These areas are too broad for you to easily standout from the pack. If, on the other hand, you become known for a niche, like [Xamarin.Forms](#) or JavaScript Visualisations you're much more likely to be valuable to those looking specifically for that skills set.

10. Age of Skills

Information is the specific knowledge that you need to solve problems. Skills represent the ability to implement solutions using your knowledge.

In a world where [most of the knowledge and tools are virtually free](#), what makes the difference? The skills, of course. We are no longer a knowledge-based society, we are a skills-based society. There was a time when almost all university degrees guaranteed a good job. Now this is no longer the case. Nobody cares about what you know. People care about what you can do. [They pay you to do things, not to know things.](#)

Teach yourself programming in 10 years.

Researchers have shown it takes about ten years (or 10 000 hours) to develop expertise in a field.

The solution is reflective practice: it's not enough to repeat the same things over and over again, but challenging yourself with a task that exceeds your current ability, to try it, to analyze its performances during and after, and to correct all error. Then repeat. And repeat again. It seems there are no real shortcuts. Learning through reading is good. But [getting your hands dirty in practice](#) is better. [The best type of learning is learning by doing.](#)

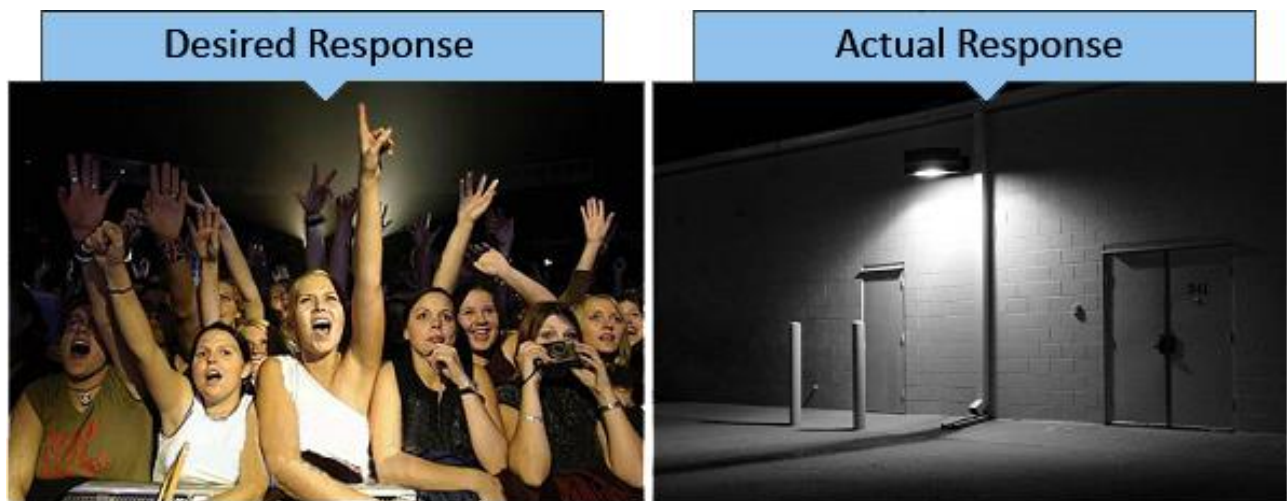
Personally, [small projects and prototypes actually helped me improve](#). But there are still interesting things to master, therefore let's keep learning.

Should you build it if nobody comes?

By [Darren Mart](#)

Not long ago I received an E-mail from an energetic and talented developer who was tackling an ambitious software project. He asked if I had any advice to pass along.

It all came flooding back. The daydreams. Springing out of bed in the dead of night because an idea can't wait. Months of painstaking work, coaxing that baby from rolling to wobbling to walking on its own. All of this for an eager audience. Except...



The cycle after the cycle

I must have drafted a half-dozen responses to his mail. I admired his enthusiasm and understood where his head was. But I also knew what likely awaited him after the arduous development cycle: another cycle of turbulence, a series of clashes between expectations and unfortunate realities.

Where did it all go wrong?

This is a dangerous question we ask ourselves as developers. "Dangerous" because it carries the assumption that we did something wrong. It's entirely possible that what you've crafted is quite good, maybe even brilliant. The issue may have nothing to do with the quality of your work, rather your criteria for success.

So how does a developer accurately gauge the success of a solo project? I wish I had a concrete answer to that. Instead I can offer a few tips on how not to gauge it:

Tip 1: Beware the social platform sirens

Ah, but they lure us in with the promise of mass consumption and acceptance. We seldom consider their ability to trample our spirits. It's possible that you'll spend months on a project, share it on Facebook with the vigor and excitement of a dog with a chew toy, and then watch the "Like" count soar to... 3. One of those came from your mother, another from a page you made to promote your project, and the other from an errant tap on a cell phone.

It's the same chilling effect you get from photos of abandoned amusement parks. You strain to imagine giggles of delight and unbridled excitement, and you struggle to accept what's staring you right in the face.

Meanwhile, your friend's profound status of "I like ravioli!" earns 23 likes and 16 comments. Popularity and substance are two different things, and we'll just have to live with that.

Tip 2: The apathy isn't personal (it might not even be apathy)

You're already aware of this but it helps to be reminded. Most people have no concept of what it takes to pull off what you've accomplished. They don't realize you single-handedly built something that'd rival the efforts of an entire team. You can't, and won't, make them genuinely care.

Apps that set the world on fire generally fall under two categories: those that help users promote themselves, and those that require little thought. If your project requires a mental investment greater than that of Candy Crush Saga, here's the harsh reality: you're gonna end up with a lot of leftovers at your launch party.

Tip 3: Think twice before soliciting feedback from your peers

In the film *Midnight In Paris*, aspiring writer Gil asks Ernest Hemingway to read his novel and offer an opinion.

HEMINGWAY: My opinion is I hate it.

GIL: You do? But you haven't even read it.

HEMINGWAY: If it's bad I'll hate it because I hate bad writing and if it's good I'll be envious and hate it all the more. You don't want the opinion of another writer. Writers are competitive.

Don't overlook the profundity of this exchange. It applies to developers, too.

Tip 4: Like it or not, you're an artist

If you pour your heart and soul into solo projects because you enjoy the creative challenge, you're an artist. If you do it because you're attempting to solve a problem, you're probably an artist-engineer hybrid. If you do it strictly for the money, you bailed on this article a long time ago so it doesn't matter what I call you.

Just like the painters and writers and decorators and gourmet chefs of the world, we secretly hope the public will be elevated and inspired by our creations.

But there's a key difference. If someone isn't inspired by a painting, they'll still be able to appreciate the individual effort. With software there's no such luxury. It doesn't matter if you're the Rembrandt of the coding world; your app is a dud compared to Office 365 or Google Maps or Skyrim, never mind the fleet of resources required by the latter.

So... should you build it if nobody comes?

Yes. Because as artists, we are enriched by the overall process regardless of the end result. Trite as it sounds, we have the sense of accomplishment and the pride of knowing how much discipline it took to see it all the way through. We stimulated our minds, we learned what does and doesn't work, we gained applicable experience. The next project, whether it's personal or professional, will reap the rewards.

Useful Job Search Websites For Programmers In The UK

By [Necemon](#)



Job boards are websites that facilitate job hunting.

There are many career websites designed to allow employers to post job requirements for a position to be filled; prospective employees can locate and fill out job applications and/or submit digital resumes for the advertised positions. Those sites may also offer employer reviews, career and job-search advice, and describe different job descriptions or employers.

As a technology specialist in the UK, here are some of the websites I found helpful in the past few years.

Graduates

[targetjobs.co.uk](https://www.targetjobs.co.uk)

Graduate jobs and schemes. Internships and placements. Great advice to help you get hired.

[eurograduate.com](https://www.eurograduate.com)

Featuring thousands of graduate careers and job opportunities across Europe.

[insidecareers.co.uk](https://www.insidecareers.co.uk)

Graduate jobs, internships, placements and school leaver schemes as well as career advice by sector.

[milkround.com](https://www.milkround.com)

The UK's most widely used student and graduate job website.

[prospects.ac.uk](https://www.prospects.ac.uk)

Prospects for graduate jobs, postgraduate study, advice about work experience, internship opportunities and graduate careers.

IT World

uk.dice.com

Formerly [The IT Job Board](#). UK Contract and Permanent IT Jobs

purelyit.co.uk

Lots of contract and permanent IT Jobs including Software Engineer, IT Director, Senior Web Developer and many more.

computerjobs.com

Specialising in Permanent and Contract IT jobs in the UK with thousands advertised daily.

cwjobs.co.uk

One of the leading UK IT job board. Search information technology jobs and apply online.

currentitjobs.co.uk

Formerly findingitjobs.co.uk. IT job board focused on providing the best IT Jobs in London.

stackjobs.co.uk

A trending IT job board in the UK with effective IT recruitment solutions.

Science and Engineering

newscientistjobs.com

Technology and science jobs, courses and career advice.

naturejobs.com

Featuring access to job listings, editorial content about scientific careers and other information.

justengineers.net

A wide range of Engineering Jobs in the UK and Worldwide.

engineeringjobs.co.uk

Latest Engineering Jobs from across London and the UK.

General Job Search Sites

reed.co.uk

One of the leading UK job sites.

monster.co.uk

Possibly the most popular job site worldwide. Resources to create a killer CV, search for jobs, prepare for interviews, and launch your career.

jobserve.com

Powerful, quick job search. Specialising in permanent and contract jobs in the UK with thousands advertised daily.

jobs.ac.uk

UK & international job search for academic jobs, research jobs, science jobs and managerial jobs

totaljobs.com

Instant job matches, alerts and more from UK companies and recruiters.

topjobs.co.uk

Searchable database of vacancies by type and region.

neuvoo.co.uk

Presumably, your job search starts here.

jobs.trovit.co.uk

Job ads from thousands of websites in just one search.

jobbydoo.co.uk

Aggregating, analyzing and listing job openings from more than 1,000 sources, including UK career sites, job boards and recruitment agencies.

Microsoft Professional Certifications for Developers

By [Necemon](#)



Microsoft offers a [wide range of online certification programs](#) designed to help you grow your skills and your career.

This article will focus on developer certifications. Microsoft Certified Solutions Developer (MCSD) is a certification intended for IT professionals seeking to demonstrate their ability to build innovative solutions across multiple technologies. For example, the [MCSD App Builder certification](#) validates that you have the skills needed to build modern mobile and/or web applications and services.

I got my first Microsoft Certification back in 2008. I have been upgrading over the years and I am now a Certified Solution Developer. Was it worth it? Sometimes it didn't matter, sometimes it was pretty useful. In my experience, here are 5 advantages of getting certified:

- Getting a Microsoft Professional Certification doesn't guarantee anything about getting a job, a raise or a promotion but it does increase the odds.
- When hiring someone new, some companies check out his certifications as well as experience. Many hiring managers verify certifications among job candidates and consider those as part of their hiring criteria. Some consider IT certifications a priority when hiring for IT positions.
- Getting a certification can boost your self-confidence, your confidence about your skills.
- It shows seriousness and passion (you did it all because you wanted to, not because you had to).
- The process of preparing for the certification increases your theoretical and practical skills.

Exam and preparation tips

My key recommendation is to dedicate a specific period to prepare right before the exam. Get some books, attend a course (virtual courses as a personal preference, but classroom courses available) and practice the technologies involved.

- Books: there are often preparation books related to the given exam. Example: [Exam Ref 70-480: Programming in HTML5 with JavaScript and CSS3](#)
- Classes : The most popular exams have short courses available online. Example: [Course 20480B: Programming in HTML5 with JavaScript and CSS3](#)
- Beyond the training material: researching the topics involved
- Actually practicing the technologies
- Answering to multi choice questions: Proceed by elimination. In case of doubt about the right answer, exclude the answers that don't make (any) sense, (or that makes less sense).

Career paths for developers

Make sure you double check that part. They keep updating the offers based on the latest technology but at the time of writing this, there are 5 options for MCS D (Microsoft Certified Solutions Developer)

- MCS D: Web Applications. Expertise in creating and deploying modern web applications and services.
- MCS D: Windows Store Apps. Expertise at designing and developing fast and fluid Windows 8 apps. There are two paths to achieving this certification-using HTML5 or C#.
- MCS D: SharePoint Applications. Expertise at designing and developing collaboration applications with Microsoft SharePoint.
- MCS D: Azure Solutions Architect. Expertise covering the full breadth of designing, developing, and administering Azure solutions.
- MCS D: Application Lifecycle Management. Expertise in managing the entire lifespan of application development.

Certification Planner

The [Certification Planner](#) is a tool to help you know what requirements are necessary to achieve your next certification. It's about planning your next steps (your options, which exams to take, in which order).

Other links of interest

- [MCS D Certification](#)
- [Web Apps certification](#)
- [Wikipedia Page](#)

That's it for now. Now get back to work.

Chapter 7

C# .NET Programming

Featuring Holty Sow



Why I like C# so much

By [Necemon](#)

If you are not [into programming](#), I need to start by telling you that a programming language is basically an artificial language (system of communication) designed to communicate instructions to a machine, typically a computer.

Now there are a lot of programming languages out there. Some are more popular than others, some are more recent, some are more powerful to some extent.

In an ideal world, each programming language serves a specific purpose. So an engineer should be able to adapt to the on going project and choose the optimal technologies. But the truth is, we very often tend to feel comfortable with some languages and find some others kind of painful. It depends on the features of the language and how long we have been using it. It's a bit like natural languages (human languages). You may learn many languages (French, Italian, Spanish, German, Japanese, etc.) and wherever you go, there will be one language that would be more relevant and that you would have to use, but as a native English speaker, you would mostly feel more comfortable speaking English than anything else. If you get in a non English speaking country, you may sure have to adapt to the local language but you would feel some relief when you meet people you can speak English with, as this is what comes naturally to you.

The difference is that you don't choose your main human language. It's generally the language they speak at the place you were born and grew up, the language your parents speak, the language your friends and teachers speak at your school, etc.

Programming languages are a different matter. Programmers [do choose to learn](#) a language, even though their motivations may be different. What I want to discuss here is, [why and how people choose their programming languages](#) ? What's the best way to go about it ?

From what I observed, there are 2 main reasons people go for a specific language:

- social proof: I guess this concern mostly the beginners, when you want to learn programming for the first time, you don't know much about the languages but you got to start somewhere. So you just go for the language that's popular among your friends, your lecturers/mentors or at your school/college. In short, you go for the languages you are most exposed to or you take advice from people around you. The upside is that you would definitely be surrounded with people who are into the same technologies so they will be able to guide and support you, work along with you on same projects. It's a bit like choosing to buy a specific video game console because all your friends have the same. Suppose you get stuck at some level, there is probably one of your friends that can tell you what to do. Plus you can exchange games with them, discuss game news & cheat codes, enjoy playing together. In short, you become part of the community and it goes pretty much the same when it comes to choosing a language.

- project requirement: [throughout their career](#), developers come across many projects they are compelled to do. The requirements may lead them to learn other languages and technologies, for example, if you work for some company and they assign you to work on that new Python project, you would need to learn Python. Even if you are an independent developer, you may learn another language as it becomes popular with your customers or as it better meets the needs of your users (in terms of speed or user experience for example).

Personally, I think both reasons are valid points. Regarding the first reason, I would just add that [it's not about following the trend just for the sake of it](#). You should make some research to find out which language will help you better do what you are trying to do. Regarding the second reason, I realise it is required to do things that you don't really like but, as often as possible, I would advise you to use the technologies and languages that you like better. If programming is your job, you better enjoy it. [Do what you love](#), really.

[Coming to my own experience](#), I guess I went for C#.NET firstly because it is very popular in the institution where I started getting serious about programming (NIIT). But this is only the reason why I got introduced to C#. There are many other reasons that keep me going.

The main reason being that I find it comfortable but not just because I had been using the most that time. Anyway it wasn't my first programming language. I did program in VB and Pascal before, so it's not that I got stuck to the first language I liked to use. By comfortable, I mean that I enjoy writing C# code. Personally, I would rather write code if I have a lot of fun doing it than write code in some obscure language that's just painful. It's true that we need to worry about things like performance and user experience but I believe that one should enjoy what they are doing.

Now what's so cool about C# ? I won't be discussing how C# compare to other programming languages and whether it's technically better or not. That's not the point of this article and as I said before there is no perfect language, there is just a right language for the right circumstances. I will just say what make C# so awesome to me (and probably to you as soon as you give it a try):

I can't help it, I need to mention that Visual Studio, the work environment for C#, is probably the world best IDE. Automated features like IntelliSense and controls drag and drop save a lot of time and effort. It's not being lazy, it's really about productivity. Whether you work solo or as a team, you always have some important tasks that are not necessarily programming oriented. Automation helps you avoid spending time and effort on the obvious, frequent, basic tasks and focus on the things that are most important.

[The language itself is easy to grab](#), it follows the same kind of syntax that C, C++, Java, etc. So anyone with a similar background can quickly get going. Also, it's known to be simple and elegant.

C# is powerful. It helps me make ANYTHING I want, whether it's a website, a web application, a web service, a smart client application, a game, a windows application, a windows service or a in-browser (Silverlight) application, etc. While most of the languages are used just for a specific purpose, either for the

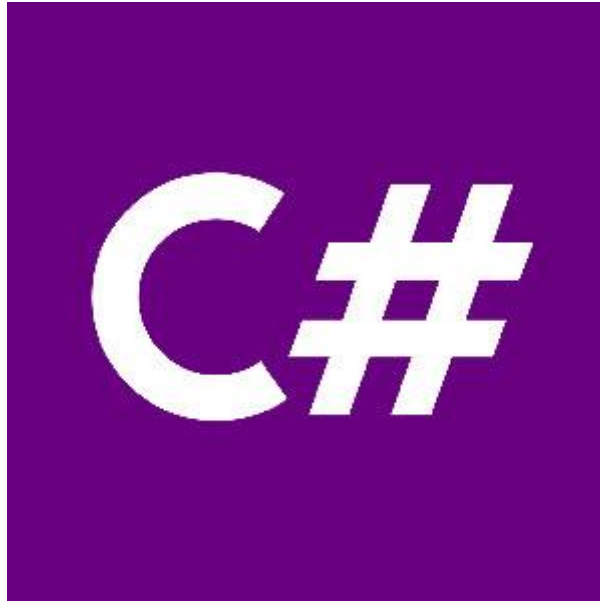
web server, for the client or for the browser, [C# does it all](#). The obvious advantage is that you don't have to learn a new language when you want to start a new kind of application.

Interoperability and language integration: C# applications and services can talk to each other. In fact, they can communicate with any other .NET applications. For example, you can easily make a WPF C# application get data from a WCF VB.NET service and pass it to a Silverlight C# application.

Well, now you know why I like C# so much...

C# Free Learning Resources For Beginners And Professionals

By [Necemon](#)



[C# \(C Sharp\)](#) is Microsoft [flagship programming language](#), hence its [popularity in the tech industry](#). Used by a large and growing number of professionals, it helps when building all kinds of applications.

A frequently asked question is, where do I start with C#, what are the best free courses?

Here are a few suggestions.

[Microsoft Virtual Academy](#)

C# fundamentals for absolute beginners. Step through 24 practical and easy-to-understand C# training episodes. Tune in to learn the basics of the C# language, and learn to apply them in your programming endeavors, like video games, mobile environments, and client apps.

[learncs.org](#)

Free interactive C# tutorial. Whether you are an experienced programmer or not, this website is intended for everyone who wishes to learn the C# programming language. There is no need to download anything, just click on the chapter you wish to begin from, and follow the instructions.

[Solo Learn](#)

The best way to learn to code is to code. Gain an understanding of C# concepts by going through short interactive texts and follow-up fun quizzes. Their beautifully designed code editor lets you make changes to existing code or write and run your own custom code and see the output on your mobile device. You can code while going through the core lessons or as a stand-alone learning activity. The more you play, the better you get!

[Codeasy](#)

A free interactive online course for learning to program C# language. It is designed for absolute beginners and does not require any prior knowledge to start. It is really fun to learn with Codeasy just by reading an adventure story about fighting machines in the future. While reading, the user meets challenges, which require real coding to solve. User can write code directly on the website. The course consists of chapters, like a real book. Each chapter has several tasks to solve by coding. The final goal is to become a programmer and to save the world.

[Visual Studio Dev Essentials](#)

Free tools and free training. Free access to technical training from industry leaders such as [Pluralsight](#), Wintellect, and [Xamarin University](#).

[Channel 9](#)

A Microsoft community site for Microsoft customers. It hosts video channels, discussions, podcasts, screencasts and interviews. This includes courses like C# Fundamentals for Absolute Beginners.

Bonus : Top Youtube playlists and tutorials on C#

[Youtube 1](#)

[Youtube 2](#)

[Youtube 3](#)

[Youtube 4](#)

[Youtube 5](#)

How to Track Mysterious Bugs with Visual Studio

By [Necemon](#)

[Debugging](#) is the process of finding and resolving defects or problems within the program that prevent correct operation of computer software or a system. Debugging tactics can involve interactive debugging, control flow analysis, unit testing, integration testing, log file analysis, monitoring at the application or system level, memory dumps, and profiling.

Some bugs may be easy to track. I don't want to talk about those. Let's discuss the really mysterious ones.

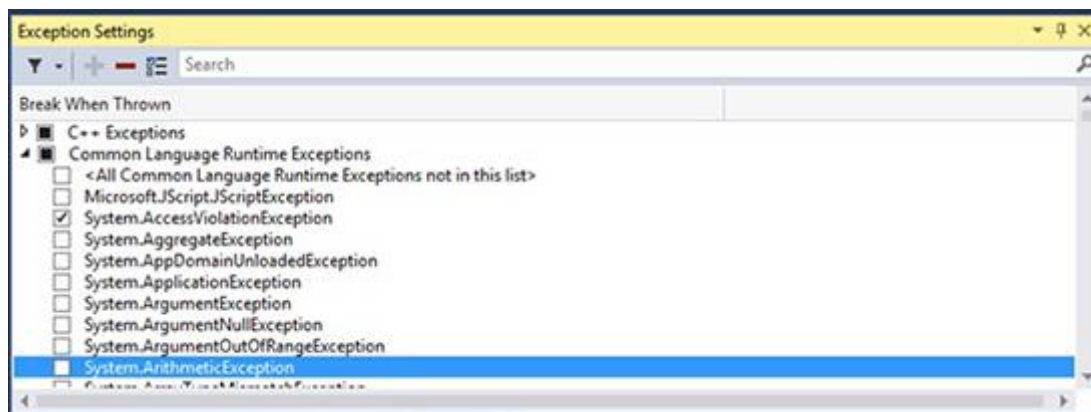
Like the saying goes, Theory is when one knows everything but nothing works; Practice is when everything works but nobody knows why. Sometimes Theory and Practice go hand in hand: nothing works and nobody knows why. In such cases, it's worth remembering about some [effective techniques](#) that often allow us to reach the root of the problem and to display its finer details. [Over time](#), here are the 3 techniques that have been most useful when debugging with Visual Studio:

1. Managing relevant exceptions with the Exception Settings Window

An [exception](#) is an indication of an error state that occurs while a program is being executed. You can and should provide handlers that respond to the most important exceptions, but it's important to know how to set up the debugger to break for the exceptions you want to see. You can use the Exception Settings window to specify which exceptions (or sets of exceptions) will cause the debugger to break, and at which point you want it to break. You can add or delete exceptions, or specify exceptions to break on. Open this window when a solution is open by clicking **Debug / Windows / Exception Settings**.

You can find specific exceptions by using the Search window in the Exception Settings toolbar, or use search to filter for specific namespaces (for example System.IO). The debugger can break execution at the point where an exception is thrown, giving you a chance to examine the exception before a handler is invoked.

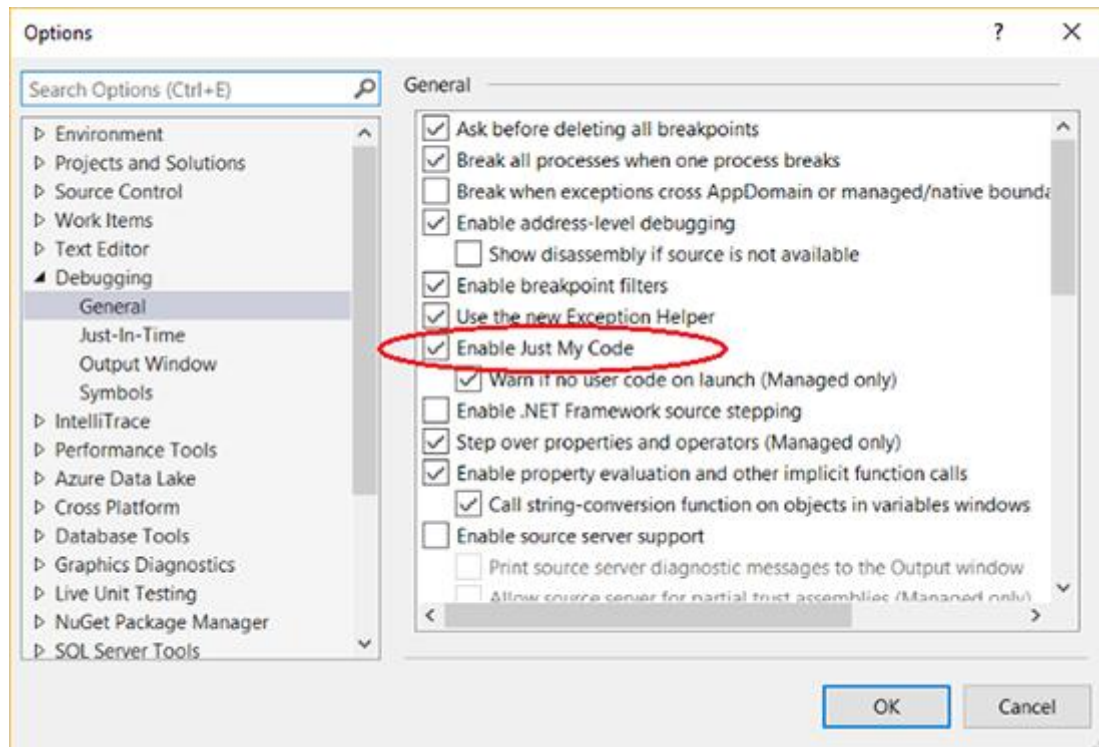
In the Exception Settings window, expand the node for a category of exceptions (for example, Common Language Runtime Exceptions, meaning .NET exceptions), and select the check box for a specific exception within that category (for example System.AccessViolationException). You can also select an entire category of exceptions.



2. Not "Just My Code"

By default, the Visual Studio debugger only breaks on exceptions generated from your own (user) code, hence skipping other system, framework, and other non-user calls. The feature that enables or disables this behavior is called "Just My Code". Depending on what you are debugging, you may want to disable it, because the source or description of the issue might well be outside of "your" code.

To disable (or enable) Just My Code, choose the **Tools > Options** menu in Visual Studio. In the **Debugging > General** node, clear (or choose) **Enable Just My Code**.



3. Recommended tools for Tracing and Error Logging

Sometimes you need to record and analyse the full details of the errors, the events and the inner exceptions: that **tracing** involves a specialized use of logging to record information about a program's execution, typically for debugging purposes. Here are my favorite logging and tracing tools:

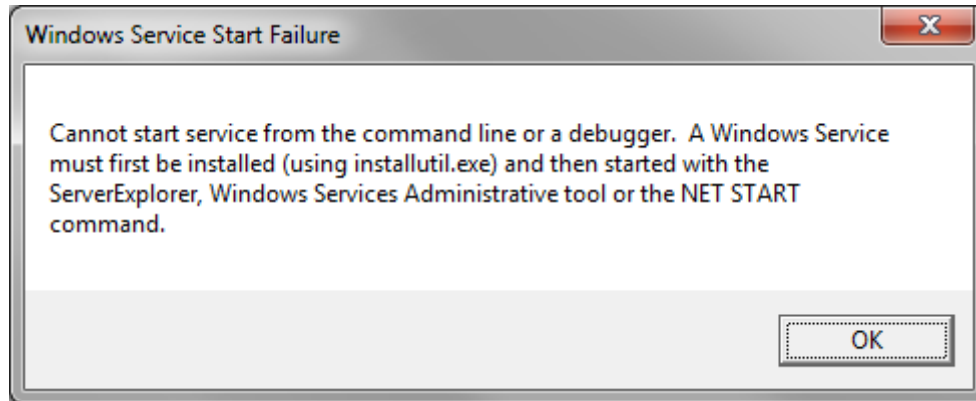
- [Systems.Diagnostics](#)
- [Microsoft Enterprise Library](#)
- [NLog](#)
- [Elmah](#)
- [Log4net](#)

Voilà.

How to Debug a Windows Service in Visual Studio

By [Holty Sow](#)

When you create a Windows Service project under Visual Studio, you may have noticed that the following dialog box appears when you try to run the service:



In summary, it is simply impossible to run a **Windows Service** in Visual Studio, you must necessarily go through the **NET START** command to start the service after having previously installed it with the command **INSTALLUTIL**. Except that this method prevents us from easily debugging the Windows Service. As a matter of fact, to debug our code, we must install the service, start it, then link the debugger to the relevant service process from Visual Studio. And by the way, let's keep in mind that we will also have to stop, recompile and restart the windows service in order to load up any change we make to the code. In short, it's kinda annoying :D

There is a simpler solution though. We can formulate different compilation directives to detect which mode we are running: **RELEASE** or **DEBUG**. If we are in:

- **DEBUG** mode: we will treat our windows service as a simple client application by displaying a dialog box indicating that the service has started
- **RELEASE** mode: the operation will remain the same as when we tried to run our Windows Service under Visual Studio. In other words, this mode should be used in production, after the debugging session is complete

Our use case will be very simple, we will just be running a Windows service and debugging the WCF service that it hosts. Here are the steps you need to follow:

1. Tweaking the code of the windows service: we will add two methods, **StartWCFService** and **StopWCFService**, which should start and stop listening to WCF incoming requests respectively. The first method will be called in the **OnStart** method definition, and the second one will come from the **OnStop** method. Below is the code:

```

private ServiceHost host;

public MyWindowsService()
{
    InitializeComponent();
}

protected override void OnStart(string[] args)
{
    StartWCFSERVICE();
}

protected override void OnStop()
{
    StopWCFSERVICE();
}

public void StartWCFSERVICE()
{
    host = new ServiceHost(typeof(IWCFSERVICE));
    host.Open();
}

public void StopWCFSERVICE()
{
    if (host != null && host.State == CommunicationState.Opened)
        host.Close();
}

```

2. Tweaking the **Program.cs** file: it's in the **Main** method that we would detect our selected compilation mode. If we are in **RELEASE** mode, then the windows service is running as usual and if we try to run it in Visual Studio within this mode, we will obviously get the so-called dialog box. If, on the other hand, we are in **DEBUG** mode, then it gets easier, we directly call the **StartWCFSERVICE** method of the instance of our windows service (so the **OnStart** method will not be called) and then we display a dialog box to notify the user. When they close the dialog box, the **StopWCFSERVICE** method is called to stop the WCF service (therefore, the actual **OnStop** method of the Windows service will not be called).

Here is the code of the **Main** method:

```
static void Main()
{
    MyWindowsService service = new MyWindowsService();
    #if DEBUG
        service.StartWCFSservice();
        MessageBox.Show("Le service a démarré...");
        service.StopWCFSservice();
    #else
        ServiceBase[] ServicesToRun;
        ServicesToRun = new ServiceBase[]
        {
            service
        };
        ServiceBase.Run(ServicesToRun);
    #endif
}
```

I hope this post has been helpful ;-)

My Essentials : Top 12 Tech Courses on Pluralsight for .NET Developers

By [Necemon](#)



Pluralsight is the largest online tech and creative library on the planet: an online education platform that offers a variety of video training courses for software developers, IT administrators, and creative professionals through its website.

I have been learning quite a bit from Pluralsight, and here are my favorites so far.

[Learning Technology in the Information Age](#)

So much to learn, so little time... This course will show both beginners and experts how to learn more in less time.

Would you write code without a design? Build hardware without a schematic? Configure a server without a plan? Of course not. Yet, how many of us learn technology without any planning, choosing resources at random in the hope that one of them will be worthwhile? This is horribly inefficient, and worse, can leave critical holes in your knowledge and skills. In this course you'll learn how to design a plan for learning any technology effectively and efficiently based on your own needs and goals.

[Reprogramming the Developer Mind](#)

Explore the habits and career tactics that create remarkable developers. Take control of your career, and set yourself apart from the pack. Become an Outlier.

This course is about making a paradigm shift in how you manage your career. We'll discuss concrete activities and skills that transform average developers into outliers. You'll learn why developers can't afford cable, ways to improve your "luck surface area", and techniques to compress your career through accelerated development. You'll learn the foundational skills for becoming an outlier: command your time, hack your image, and own your trajectory. Prepare to think about your development career in a whole new way.

[Encapsulation and SOLID](#)

This course teaches how to write maintainable and flexible object-oriented code. Learn how to write maintainable software that can easily respond to changing requirements using object-oriented design principles. First, you'll learn about the fundamental object-oriented design principle of Encapsulation, and then you'll learn about the five SOLID principles, also known as 'the principles of object-oriented design.' While this course is aimed at beginner to intermediate developers, it's based on decades of experience, so even advanced programmers can learn a thing or two. There are plenty of code examples along the way; while they're written in C#, they should be easily understandable to readers of C, C++ or other curly-brace-based languages.

[Date and Time Fundamentals](#)

This course will help you to understand dates and times, and how they should be used in software development.

Managing dates and times properly is one of the most difficult things to get right in software. This is mostly due to how humans have introduced nuance into our calendars and clocks. In this course, you can straighten it all out. You will learn about UTC, daylight saving time, time zones, and calendar systems. You will also learn how date and time values are represented and manipulated in various programming languages. We will look closely at the different kinds of time zone data, and discuss various fallacies and gotchas that are commonly encountered. We will deep dive into how date and time are handled in the .NET Framework, and in JavaScript. We will also look at various libraries that make things slightly more bearable. Throughout the course, you will learn about real-world situations that require deeper thought about how date and time are handled in your applications, and I will give you practical advice on how to solve them.

[Web Security and the OWASP Top 10: The Big Picture](#)

Security on the web is becoming an increasingly important topic for organisations to grasp. This course takes you through a very well-structured, evidence-based prioritisation of risks and most importantly, how organisations building software for the web can protect against them.

[Visual Studio : Essentials to the Power-User](#)

This course introduces Visual Studio and includes productivity boosters for everyone to make writing and reading code easier and more fun. Visual Studio is an integrated development environment you can use to create applications and libraries with many different frameworks and languages. It has a rich feature set, including an intelligent editor, built in compiler (and related tools), and context sensitive help. This course starts with basic concepts like projects and solutions, shows you how to make Visual Studio look and work the way you want it to, and demonstrates how to use the most popular tool windows and dialogs. It goes further into tips and shortcuts that will save you time every day. Using Visual Studio is about more than writing code or reading code written by others. To be truly productive, you need to debug well and understand the designers that help you build your user interface. This course also shows you how to add helpful extensions that make Visual Studio even better. When you've completed it, you'll know how to use the tool itself and can focus on a specific language or framework as your next step.

[Bootstrap 3](#)

Twitter's Bootstrap 3 can help you achieve a great looking and performing web site. Building great looking websites that work well with different sized devices can be a challenge. By utilizing Bootstrap 3 framework, you can meet that challenge head-on. Bootstrap 3 is a mobile-first responsive design framework for structuring your website's HTML. It includes a great grid system, responsive design, CSS typography and components to solve many of the most common design challenges that face web developers today.

[Building Your First Xamarin.Android App from Start to Store](#)

[Xamarin is a cross-platform development tool](#). It solves dilemmas many developers face when developing cross-platform apps: separate coding languages and UI paradigms. With Xamarin, you can use C# for iOS, Android, and Universal Windows apps.

In this course, you'll learn how to build a complete, fully-working Xamarin.Android app using C#. Xamarin.Android is covered in a very practical, hands-on way that you can follow along.

[More Effective LINQ](#)

Learn how to fully harness the power of LINQ by exploring best practices and avoiding common pitfalls by solving some fun and challenging problems. In this course, you will learn how to take full advantage of the power and capabilities of LINQ. You will see how the LINQ extension methods can be combined together to solve complex problems in a simple and succinct "pipeline." Throughout the course, you will learn to solve some "LINQ Challenges" and pick up lots of pro tips that will take your LINQ skills to the next level, including how to extend, debug, optimize, and test LINQ.

[Async C#](#)

In this advanced series, Jon Skeet shows us the new Asynchronous goodness available in C# 5.0. Managing threads, awaiting asynchronous calls - these are all made simple with a few new keywords and types. Asynchronous coding in a static language is not exactly "simple," and rather than try to simplify complex topics, Jon have, instead, decided to go deep and see how things work at a deeper level. Again, this is an advanced production! If you are new to C# (or new to programming in general), you may want to become familiar with the basics of C# before you tackle this one.

[TypeScript Fundamentals](#)

TypeScript is an open source language that provides support for building enterprise scale JavaScript applications. Although several patterns exist that can be used to structure JavaScript, TypeScript provides container functionality that object-oriented developers are familiar with, such as classes and modules. It also supports strongly-typed code to ensure inappropriate values aren't assigned to variables in an application.

In this course, John Papa ([link to his site](#)) will walk you through the key concepts and features that you need to know to get started with TypeScript, and use it to build enterprise scale JavaScript applications. You'll learn the role that TypeScript plays as well as key features that will help jump-start the learning process.

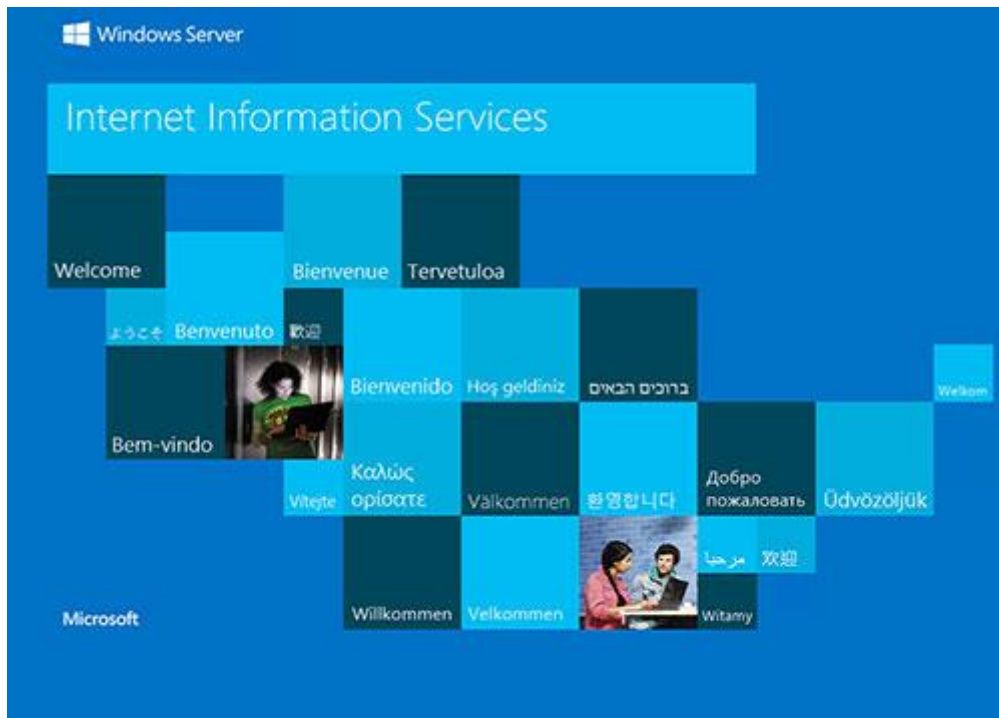
[Understanding ASP.NET Core](#)

This course will help you quickly grasp how to get going with ASP.NET Core with a compact, practical presentation, covering all the best new features of ASP.NET since its release.

This latest release of .NET has a lot to be excited about, it not only supports development for a lightweight version of .NET called .NET Core, but for the first time you can also target non-Windows platforms. In this course, Understanding ASP.NET Core, you'll get to see what concepts are most important to get you quickly up to speed. You'll start with seeing what's changed within the project structure and mechanics in Visual Studio. Then, you'll get to see what's new in MVC for ASP.NET Core along with sharing code across existing .NET frameworks and newer .NET Core frameworks. Finally, you'll get to see it's versatility first-hand when an application is deployed to multiple operating systems. By the end of this course, you'll be ready to take advantage of the best new features of this latest release of .NET.

Deploying a Web App to IIS / Windows Server In 7 Clever Steps

By [Necemon](#)



Here is a simple deployment routine, for a reliable and smooth release :-)

1. Create/configure [the website](#) and [its application pool](#)
2. Copy the files over : [setup an FTP\(S\) connection](#), as well as the client access ([FileZilla](#), [Visual Studio](#), etc.)
3. [Assign writing permissions](#) to the application identity on the relevant folders (for logging, files uploads, etc.)
4. Set time outs to appropriate values
[Idle Timeout](#): you can change it from the default of 20 to however many minutes you want. You can also adjust the setting to 0 (zero) which effectively disables the timeout so that the application pool will never shut down due to being idle. This can be configured in the Advanced Settings of the application pool.
[ConnectionTimeOut](#): specifies the time (in seconds) that IIS waits before it stops a connection that is considered inactive. This can be configured in of the Advanced Settings of the Administrative Tools (system.applicationHost/weblimits).

5. Configure Auto-Start

A common problem is the need to perform initialization tasks and "warm up" tasks for a web application. Larger and more complex web applications may need to perform lengthy startup processing, prime in-memory caches, generate content, etc... prior to serving the first HTTP request. One way to fix this is to twist a couple of properties in the [application initialization module](#):

- Set the application pool StartMode property to AlwaysRunning
- Set PreloadEnabled to true, and specify the application pool and path

6. Configure SQL Server / Set up automatic data Backups

Create a [maintenance backup plan with SQL Agent](#)

7. HTTPS / SSL [Certificate Installation](#)

HTTPS provides [security, identity, SEO, access to HTML5 powerful features and more](#).

Launching Your Application At Windows Startup Without Hacking The Registry

By [Holty Sow](#)

In this post, I will show you two methods to configure your .NET applications so that they would launch as soon as Windows starts. These two methods do not require any change to the registry, hence you don't need to worry about cleaning up that database if the user uninstalls your application.

First method:

From the Windows Installer Deployment Project, follow these steps:

In the **File System** of the project, add a special folder called **User's Startup** Folder.

Then a shortcut of the project output is created in the Application Folder and renamed to give it a more meaningful name.

We cut (CTRL + X) the shortcut we just created and paste it (CTRL + V) in the User's Startup Folder.

Second method:

The other solution is to trigger an automatic start from within the code of the application. To do this, I created an activation function, and another one for deactivation.

N.B.: For this code to work, you will need to add a reference to the **Windows Script Host Object Model** assembly in your project.

The activation function can be written as follow:

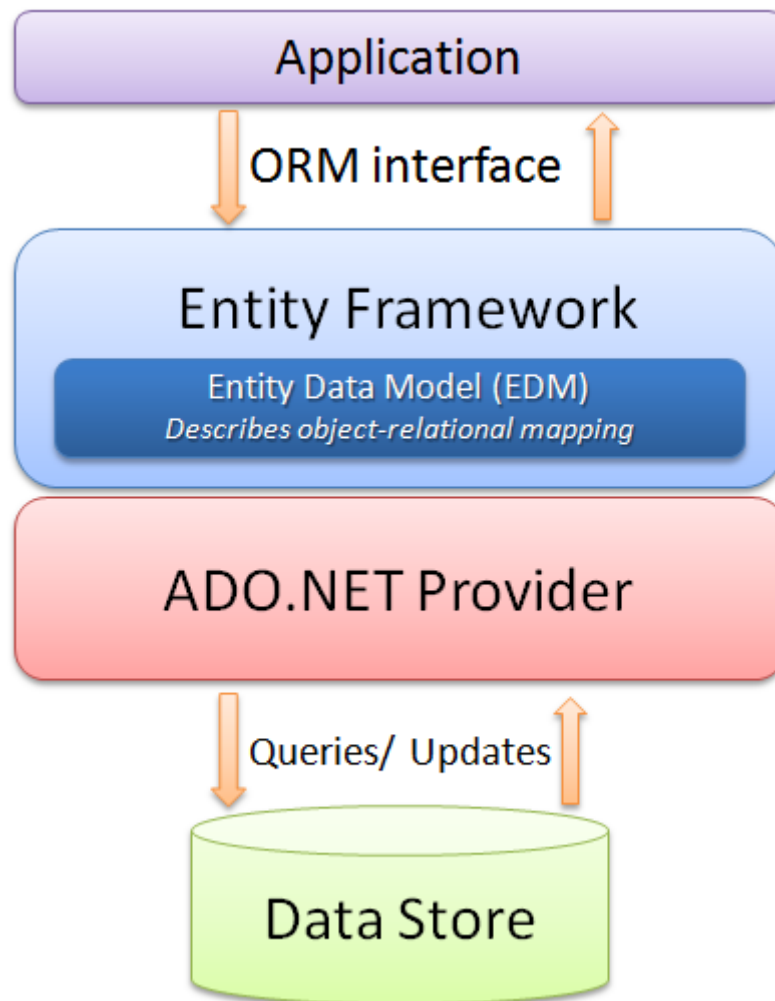
```
public void EnableApplicationStartup()
{
    string shortcutPath =
System.IO.Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.Startup),
    "MyShortcut.lnk");
    if (System.IO.File.Exists(shortcutPath)) return;
    WshShell wshShell = new WshShellClass();
    IWshShortcut shortcut = (IWshShortcut)wshShell.CreateShortcut(shortcutPath);
    shortcut.TargetPath = Assembly.GetEntryAssembly().Location;
    shortcut.Description = "My first shortcut";
    shortcut.Save();
}
```

Here is the deactivation function:

```
public void DisableApplicationStartup()
{
    string shortcutPath =
System.IO.Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.Startup),
"MyShortcut.lnk");
    if (!System.IO.File.Exists(shortcutPath)) return;
    System.IO.File.Delete(shortcutPath);
}
```

My 12 Favorite Entity Framework Tricks

By [Necemon](#)



The Entity Framework is an ORM system, a set of technologies that support the development of data-oriented software applications. It's part of the .NET Framework. I have been playing with it for a couple of years and here are my top tips :

1. Extending the TimeOut value

When loading large amounts of data, the operations happen to fail every time they hit the default time out, so it may be a good idea to extend it to a few hours.

Doing it when initializing the Context object ensures that it's all set before any database call.

2. Second Level Caching

That's a caching of query results. The results of SQL commands are stored in the cache, so that the same SQL commands retrieve their data from the Cache rather than executing the query again against the underlying provider. This can have a performance boost for your application and results in less activity against your database. To enable this feature, you can [download and implement this open source project](#).

3. Dynamic Code First and database migration

EF allows you to program against a model without having to deal with a database directly. With the Code-First approach, you can focus on the domain design and start creating classes. Code-First APIs will create/update the database on the fly based on your entity classes and configuration.

By the way, while we are on the topic on configuration, you can do it all within your C# code. Here are a couple of handy properties:

`AutomaticMigrationEnabled` : Set it to true to enable the Code-First magic

`AutomaticMigrationDataLossAllowed` : a value indicating if data loss is acceptable during automatic migration. If set to false an exception will be thrown when data loss may occur as part of an automatic migration.

4. Previewing Code-First change details

There is an easy way to generate the SQL scripts that EF plans to execute, before/without actually committing those changes. It goes like this:

```
var configuration = new MigrationConfiguration();
var migrator = new DbMigrator(configuration);
var scriptor = new MigratorScriptingDecorator(migrator);
string script = scriptor.ScriptUpdate(null, null);
```

It could be useful if you want to make some changes to that script, or if you are debugging an issue or if you are just curious about what's going on :-)

5. MARS

Multiple Active Result Sets (MARS) is a feature that allows the execution of multiple batches on a single connection. To say it in a simple way, you can make a connection to server and then submit multiple requests to server at the same time, and then read results from these requests in whatever way you want. Just include "MultipleActiveResultSets=True" in your web.config file.

6. Setting all the properties constraints in a single place

For example, instead of adding an attribute to each string property, you could set a default string maximum length as follow:

```
protected override void OnModelCreating(DbModelBuilder modelBuilder)
{
    modelBuilder.Properties<String>().Configure(p => p.HasMaxLength(360));
}
```


7. Inheritance strategy

EF has many ways of dealing with inheritance

- [Table per hierarchy](#) : a single table with all based and derived class properties (A discriminator property being used to differentiate between them)

- [Table per type](#) : base properties on the base table, and for each derived class, derived properties on a separate table

- [Table per concrete type](#) : each derived class gets its own table with all (based or derived) properties.

Pretty extensive, the only case I am not sure about : is there a way to make EF not deal with inheritance at all ? I mean, assuming A derives from B, is there a way to make EF treats A and B as completely different classes and ignore the fact that that one inherits from the other ?

Not a very common scenario, but just for the sake of avoiding duplication, I could want to use inheritance in the C# code so that I don't copy the properties twice in different classes but I wouldn't want to involve any reaction from EF. TPC comes close to that, but the classes still share the same primary key.

8. EntityFunctions Methods

When using LINQ to Entity Framework, your predicates inside the Where clause get translated to SQL, which doesn't have a translation for some complex constructions like `DateTime.AddDays()`. That's where [EntityFunctions methods come in the picture](#).

9. LINQKit

[LINQKit](#) is a free set of extensions [for LINQ to SQL and Entity Framework power users](#). I mostly use it to dynamically build predicates and insert expression variables in subqueries, but it also allow to :

- Plug expressions into EntitySets and EntityCollections
- Combine expressions (have one expression call another)
- Leverage AsExpandable to add your own extensions.

10. Lazy Loading

One of the most interesting Entity Framework features is the Lazy Load function, which is the process whereby an entity or collection of entities is automatically loaded from the database the first time that a property referring to the entity/entities is accessed.

To put it simply, lazy loading means delaying the loading of related data, until you specifically request for it. This could be enabled by setting the `Configuration.LazyLoadingEnabled` property to true.

11. AsNoTracking

Entity Framework exposes a number of performance tuning options to help you optimise the performance of your applications. One of these tuning options is [.AsNoTracking\(\)](#). This optimisation allows you to tell Entity Framework not to track the results of a query. This means that Entity Framework performs no additional processing or storage of the entities which are returned by the query.

There are significant performance gains to be had by using [AsNoTracking\(\)](#).

12. Staying away from performance traps and other EF gotchas

The object context manager can lead to situations where [EF behaves in surprising ways](#). It's good to [be aware of the patterns](#) that you can follow to [avoid these pitfalls](#).

ASP.NET MVC: Avoid Polluting The Model Of The View With Error Messages

By [Holty Sow](#)

I recently answered a question that came up on the [StackOverflow website](#). Here was the situation.

Let's consider the following model:

```
public class ForgotPasswordMV
{
    [Display(Name = "Enter your email"), Required]
    public string Email { get; set; }
}
```

This model is used in a controller action as follows:

```
[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult Search(ForgotPasswordMV viewModel)
{
    if(Temp.Check(viewModel.Email))
        return RedirectToAction("VerifyToken", new { query = viewModel.Email });
    else
    {
        ViewBag.ErrorMessage = "Email not found or matched";
        return View();
    }
}
```

The question was whether using the ViewBag dynamic property of the controller to expose the error message was a good practice, as the OP investigations suggested that it was necessary to expose properties from the model.

Obviously it is highly recommended NOT to use the ViewBag property since it does not provide any Strong Typing. If you want to communicate with the view you should always involve a typed model. The suggested solution is therefore legitimate in that respect, however, it is not a good practice when it comes to model error handling in ASP.NET MVC.

The solution for exposing error messages (as in the previous example, which does not use data annotation attributes) derives from the use of the AddModelError method, from the ModelStateDictionary class. We do not need to instantiate this class because a ModelState property containing an instance of this class already exists in the Controller.

Therefore, the right solution is the following:

```

[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult Search(ForgotPasswordMV viewModel)
{
    if
    {
        // ...
    }
    else
    {
        this.ModelState.AddModelError("Email", "Email not found or matched");
        return View(viewModel);
    }
}

```

It should be noted that this method receives as first parameter the name of the property of the model to which the error message is associated. So, in order to display this error message in the view near the Email field, you can simply add the following line next to it:

```
@Html.ValidationMessageFor(m => m.Email)
```

However it is possible to have a general error message for the whole model, that is, an error message which is not attached to any property of the model. For this, it would be necessary to use an empty String as first parameter:

```
ModelState.AddModelError(String.Empty, "Email not found or matched");
```

In the Razor View, use the following line:

```
@Html.ValidationSummary(true, "The following error has occurred:")
```

The first Boolean parameter indicates that you do not want to display error messages that are already attached to model properties.

In this post, we saw that we do not need to pollute our model, nor do we need to attach our error messages to specific properties. It's all about using ASP.NET MVC tools to make things easy.

I hope this post has been helpful to you.

Limiting The Execution Of An Action To AJAX Requests Only

By [Holty Sow](#)

In ASP.NET MVC, we manipulate Views, among other things. Some of these Views represent full pages and some are just page parts. These parts or areas belonging to a view are called partial views, and they are also returned by controller actions. Since these partial views should only be used within a view, the ASP.NET MVC framework allows us to protect any call to these partial actions by decorating them with the **ChildActionOnly** attribute. This attribute makes sure that the action:

- cannot be used as an entire view and the application developers will always run it using the **HtmlHelper.Action** or **HtmlHelper.RenderAction** methods.
- has a URL that will not be accessible via the address bar, if a user somehow becomes aware of the existence of this URL.

However, as with any dynamic site, we will have AJAX requests that can also make requests for HTML content without having to load the page completely. This content also represents a part, and when we receive the response from the web server we have to embed this piece of HTML somewhere in the page. The AJAX request sent to the server will certainly invoke a controller action. This action, like those marked with the **ChildActionOnly** attribute, must have these constraints:

- should only go through AJAX requests.
- inaccessible via the browser address bar.

But the ASP.NET MVC framework does not offer any attributes that allow us to apply these restrictions to an action, but it gives us the tools to create them. For this, we need to code a filter that will be executed just before the execution of the action in question. If the incoming request complies with the requirements of a request made in AJAX then we let the action continue its way. In case the conditions are not met, we return a 404 page (as if the URL didn't exist).

The code of the new filter that I will call **AjaxOnlyAttribute** (derived from the **ActionFilterAttribute** class) is as follows:

```
[AttributeUsage(AttributeTargets.Class | AttributeTargets.Method, AllowMultiple = false)]
public class AjaxOnlyAttribute : ActionFilterAttribute
{
    public override void OnActionExecuting(ActionExecutingContext filterContext)
    {
        if (filterContext.HttpContext.Request.IsAjaxRequest())
        {
            base.OnActionExecuting(filterContext);
        }
        else
        {
            filterContext.HttpContext.Response.StatusCode = 404;
            filterContext.Result = new HttpNotFoundResult();
        }
    }
}
```

The new class only overrides the method that we are interested in, that is, the **OnActionExecuting** method that is called just before the start of the action invoked by the incoming request. The attribute can be set on the controller to handle the set of actions where an AJAX request is mandatory.

When getting new developers onboard, they might not immediately understand why those requests keep returning **404** messages. To avoid wasting time in recurrent explanations, I think it would be worth having the **DEBUG** mode enabled by default for developers. The code of our class would then look like this:

```
[AttributeUsage(AttributeTargets.Class | AttributeTargets.Method, AllowMultiple = false)]
public class AjaxOnlyAttribute : ActionFilterAttribute
{
    public override void OnActionExecuting(ActionExecutingContext filterContext)
    {
        if (filterContext.HttpContext.Request.IsAjaxRequest())
        {
            base.OnActionExecuting(filterContext);
        }
        else
        {
            #if DEBUG
            filterContext.Result = new ViewResult { ViewName = "AjaxOnly" };
            #else
            filterContext.HttpContext.Response.StatusCode = 404;
            filterContext.Result = new HttpNotFoundResult();
            #endif
        }
    }
}
```

Since developers have to compile the application in **DEBUG** mode most of the time, that **AjaxOnly** View can provide them with some explanations and instructions. Note that this version of the class only works if an **AjaxOnly.cshtml** View has been added to the **Shared Views** directory of the application.

Alternatively, we do not have to return a partial view. We can simply replace the following code:

```
filterContext.Result = new ViewResult { ViewName = "AjaxOnly" };
```

With some simple text content:

```
filterContext.Result = new ContentResult
{
    Content = $"This action '{filterContext.HttpContext.Request.RawUrl}' was designed for  
AJAX requests only"
};
```

However, the version that uses an actual View has an advantage, it's the ability to systematically render the relevant page "**_Layout**", typically defined at the application level.

[Interview] In The Bubble Of Holty Sow: "for more efficiency, I prefer to work in an agile team"



1. Quick (but effective) introduction of the personage and her achievements?

My name is Holty Samba SOW. After having started my higher studies in Maths-Physics, then completed a professional degree in Computer Science in Senegal, I went on to further my studies in France and obtained a Master's degree in [Electronic Documents and Information Flow](#) in 2009. This allowed me to get into the professional world of computer programming. Currently I am a consultant and developer, mainly on [the .NET platform](#), at SoftFluent, which is a software publisher but also a service company.

2. What are your main goals in life?

At the moment, my main goal is to keep my passion for computer programming for as long as possible by accumulating a maximum of experiences in the Web and Mobile world. I hope to enable my native country, Senegal, to benefit from this wealth of experience.

3. What tools and techniques do you use to accomplish things efficiently?

My favorite IDEs: VS Code for anything related to CSS, JavaScript and TypeScript. For everything else, I use the Visual Studio IDE with the ReSharper extension for more efficiency.

My favorite technologies are generally anything that involves back-end technologies on the .NET platform, like ASP.NET MVC, ASP.NET Web API and recently ASP.NET Core.

The methodology I prefer to work in a team is Scrum/Agile.

4. Any recommendations for your juniors?

A book: "C# in a Nutshell" to thoroughly understand the C# programming language.

Twitter and Blogs: following [influencers \(tech speakers, CTO, etc.\) via Twitter or through their blogs](#), can help in staying aware of some good stuff. [links]

Websites:

docs.microsoft.com (I think the new Microsoft documentation website is quite well developed);

stackoverflow.com (I think that trying to answer questions from other developers can sometimes help in uncovering tips and tricks that we would not have thought of otherwise);

pluralsight.com (a lot of videos about programming, really interesting... The only catch : there's a charge for it);

channel9.msdn.com (same as Pluralsight, but it's free and the speakers are Microsoft employees).

5. What is the best way to contact you?

Email: holty.sow@gmail.com

Blog : hssow.wordpress.com

LinkedIn : linkedin.com/in/holty-samba-sow-43042715

Twitter: twitter.com/CodeNotFound

Facebook : facebook.com/CodeNotFound

The Xamarin Revolution

By [Necemon](#)



What is Xamarin?

Xamarin is a free, cross-platform development tool. It solves dilemmas many developers face when developing cross-platform apps: separate coding languages and UI paradigms. With Xamarin, [you can use C# for iOS, Android, and Universal Windows apps](#). And with Xamarin Forms, interface design for all three platforms can be accomplished within its XAML-based framework.

The Xamarin platform consists of a number of elements that allow you to develop applications for iOS and Android:

C# language – Allows you to use a familiar syntax and sophisticated features like Generics, Linq and the Parallel Task Library.

Mono .NET framework – Provides a cross-platform implementation of the extensive features in Microsoft's .NET framework.

Compiler – Depending on the platform, produces a native app (e.g. iOS) or an integrated .NET application and runtime (e.g. Android). The compiler also performs many optimizations for mobile deployment such as linking away un-used code.

IDE tools – The Xamarin Studio IDE and the Xamarin in Visual Studio allow you to create, build and deploy Xamarin projects.

In addition, because the underlying language is C# with the .NET framework, projects can be structured to share code that can also be deployed to Windows Phone.

Here is [some documentation I would recommend](#) if you consider getting started with Xamarin :

[Developer Home Page](#)

[Cross Platform Guide](#)

[10 minutes introduction for Android](#)

[Android Guide](#)

[10 minutes introduction for iOS](#)

[iOS Guide](#)

[The Xamarin Show](#)

[Xamarin University](#)

[Pluralsight courses](#)

[Lynda Courses](#)

Top 8 Tricks with Xamarin Programming

By [Necemon](#)



If you don't know what Xamarin is, may I suggest that you start by reading my [introductory notes](#) before proceeding with this?

For those in the now, let's get into it without further ado:

Xamarin.Forms

Xamarin.Forms is a cross-platform UI toolkit that allows developers to easily create native user interface layouts that can be shared across Android, iOS, and Windows Phone. You can build native UIs for iOS, Android and Windows from a single, shared C# codebase. This means that applications can share a large portion of their user interface code and still retain the native look and feel of the target platform. Xamarin.Forms allows for rapid prototyping of applications that can evolve over time to complex applications. Because Xamarin.Forms applications are native applications, they don't have the limitations of other toolkits such as browser sandboxing, limited APIs, or poor performance. Applications written using Xamarin.Forms are able to utilize any of the API's or features of the underlying platform, such as (but not limited to) CoreMotion, PassKit, and StoreKit on iOS; NFC and Google Play Services on Android; and Tiles on Windows. In addition, it's possible to create applications that will have parts of their user interface created with Xamarin.Forms while other parts are created using the native UI toolkit.

Custom Renderers

Xamarin.Forms user interfaces are rendered using the native controls of the target platform, allowing Xamarin.Forms applications to retain the appropriate look and feel for each platform. Custom Renderers let developers override this process to customize the appearance and behavior of Xamarin.Forms controls on each platform.

Custom renderers provide a powerful approach for customizing the appearance and behavior of Xamarin.Forms controls. They can be used for small styling changes or sophisticated platform-specific layout and behavior customization. This article provides an introduction to custom renderers, and outlines the process for creating a custom renderer. Xamarin.Forms Pages, Layouts and Controls present a common API to describe cross-platform mobile user interfaces. Each page, layout, and control is rendered differently on each platform, using a Renderer class that in turn creates a native control (corresponding to the Xamarin.Forms representation), arranges it on the screen, and adds the behavior specified in the shared code.

[SQLite](#)

Most applications have some requirement to save data on the device locally. Unless the amount of data is trivially small, this usually requires a database and a data layer in the application to manage database access.

SQLite is an in-process library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine. The code for SQLite is in the public domain and is thus free for use for any purpose, commercial or private. SQLite is the most widely deployed database in the world with more applications than we can count, including [several high-profile projects](#).

iOS and Android both have the SQLite database engine "built in" and access to store and retrieve data is simplified by Xamarin's platform.

[ACR User Dialogs](#)

A cross platform library that allows you to call for standard user dialogs from a shared/portable library : Actionsheets, alerts, confirmations, loading, login, progress, prompt, etc.

[Xlabs](#)

Xamarin Forms Labs is a open source project that aims to provide a powerful and cross platform set of controls and helpers tailored to work with Xamarin Forms : AutoCompleteView, Calendar Control, ImageButton, HyperLinkLabel, etc.

[UI Sleuth](#)

A Xamarin.Forms debugging tool and UI inspector. If you've ever made a web site, it's similar to Microsoft's F12 tools or Chrome Developer Tools. You can use it to efficiently track down layout issues, prototype a new design, and remotely control a device.

[GenyMotion Android Emulator](#)

Fast and easy Android emulator. The most powerful Android emulator for app developers and testers. It's available for free or premium on Windows, Mac and Linux.

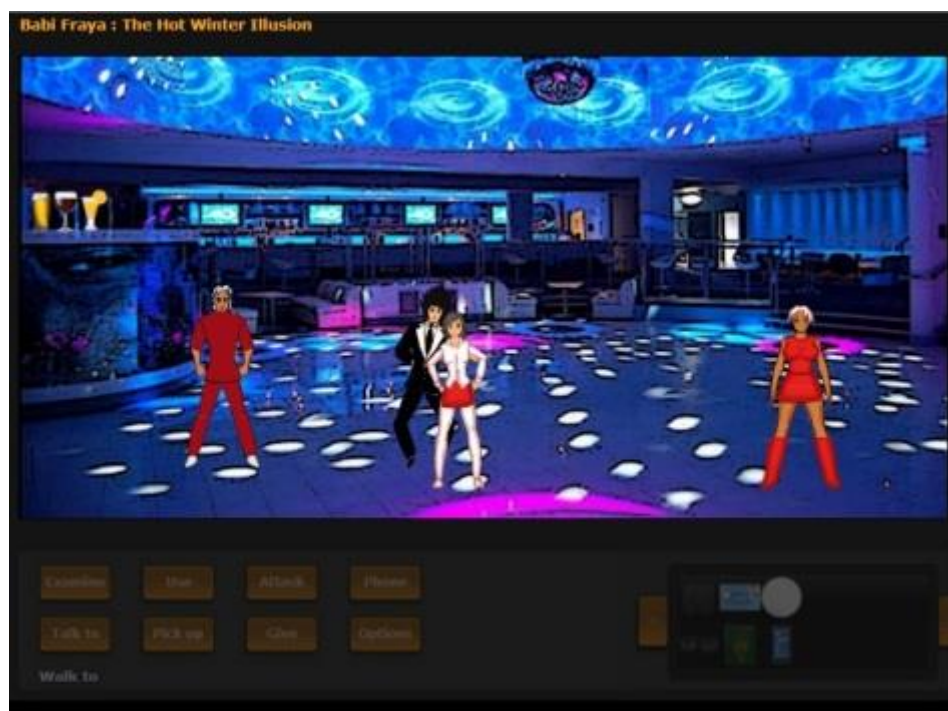
[Remoted iOS Simulator](#)

To test and debug iOS apps within Visual Studio on Windows. Most modern Windows computers have touch screens, and the remote iOS simulator lets you touch the simulator window to test user interactions in your iOS app. This includes pinching, swiping, and multiple-finger touch gestures - things that previously could only be easily tested on physical devices.

Chapter 8

Epic Prototypes, Classic Projects, Historic Genre

necemon.com | babifraya.com | evasium.com | kpakpatoya.com



Inside Bavardica - Part 1 : Discussion of the subject areas

By [Necemon](#), 2010

A modern instant messenger is composed of various features which include public conversations, private conversations, user picture and signature. When it's time to innovate and make an instant messenger that is so special that it is both interesting and standing apart from whatever program already available on the market, it is not too obvious to figure out what extra features could be attached. It feels like almost everything had been already thought of and implemented in some other application.

However, [Bavardica](#) is my attempt to make a communication platform like no other. Bavardica, derived from Bavardage, which actually means talking in French, has got mostly two assets over regular instant messengers. Firstly, there is the permanent presence of artificial beings that interact with other users. Secondly, the application relies on a two-dimensional space which represents a small world in which the users are inhabitants represented by animated characters. This project is a good chance to set the scene of what could become an innovative virtual world.



Many areas of web entertainments are involved in the Bavardica Project. In its initial form, Bavardica was basically a chat application, but as it is evolving, Bavardica also fits in the following fields.

Artificial Intelligence

As taught to me by [Dr Phil Grant](#), the main objective of AI (Artificial Intelligence) is to try to make computers perform tasks that humans tend to be good at. The actual name “Artificial Intelligence” was coined by John McCarthy in the 60’s. There are a lot of topics which might nowadays be considered as part of the general subject of Artificial Intelligence. But for our current purpose, we shall only consider the themes of natural language understanding and representation of knowledge.

Now, what does it mean for a machine to be intelligent? Alan Turing thought about this problem in the 50’s and came up with the following test: A human would ask questions to both a computer and a human in some other rooms; if the interrogator is not able to figure out (from the answers) which is the machine then the computer is said to behave in an intelligent manner. A very early attempt to build a program which would pass the Turing test was the ELIZA program (by Joseph Weizenbaum). ELIZA was mimicking a conversation with a psychologist and was quite impressive at first glance. However, it was missing a common sense and an actual understanding of natural language (which are both pretty hard to acquire from a computer point of view). Nevertheless, many programs of this kind followed. They were based on tricks like:

- o The repetition of user’s statements verbatim (subject to pronominal adjustments)
- o Preceding the repeated statement with the introductory “Why do you need to tell me”
- o Asking “Why do you ask that”, which, in effect, change the topic or the level of the conversation.

Later on, Colby and AI wrote a program called PARRY based on ideas in ELIZA which had more complex properties like elements of emotion and more knowledge of grammar. It was intended to mimic a paranoid patient who believed he was being persecuted. In a real experiment, 33 psychiatrists were asked to rate the degree of paranoia of patients who were communicating through typed transcripts. Three of them were from human patients and two generated by PARRY. None stated that they thought there was anything unusual. Later the psychiatrists were told that some of the transcripts were machine generated, but they were, on the whole, unable to recognise which ones were from PARRY. Of course this whole situation is a bit odd anyway, as human paranoids behave in strange ways. The point here is that it may be difficult to differentiate a human talk from a machine talk (Phil Grant).

Moreover, throughout the time, AI systems are becoming more and more advanced, in fact, more intelligent. Currently, one of the strongest programs of this type is ALICE (Artificial Linguistic Internet Computer Entity) by Richard Wallace. According to Wikipedia, that’s a program inspired from ELIZA that engages in a conversation with a human by applying some heuristical pattern matching rules to the human’s input. The program uses an XML schema called AIML (Artificial Intelligence Markup Language). I use a similar model for implementing the AI feature in this project.

Virtual worlds:

A virtual world can be defined as a synthetic reality, an environment that is completely generated by a computer system and that is completely transportive in that the participant controls an avatar immersed in an artificial world, according to the book *Networking and Online Games* (Armitage, Claypool and Branch, 2006). It says that virtual worlds are intended for his users to inhabit and interact. Sometimes, the user can also manipulate elements in the said world and therefore experience a kind of telepresence. As the rate of people using virtual worlds is increasing by 15% every month and as there are nearly 600 million people worldwide registered in virtual world today (according to K Zero, a virtual word consultancy service), I felt that it would be a good plan to learn about this trend and give it a try. In its current form, Bavardica is a virtual world in the sense that each bavard (avatar) is aware of the other bavards and of the scene boundary.

Multimedia:

In a web browser, operations like timing, streaming, playing music forth and back, playing a list of music files etc. are not at all trivial using basic html and javascript. One big advantage of a browser plugin like Silverlight is that it gives much more ease about operations on media files. The book 'Accelerated Silverlight 3' had been really valuable by teaching me how the `System.Windows.Controls.MediaElement` control provides media playback capability. That control can handle both audio and video in a variety of formats like mp3, wmv, asx, etc. There is even a support for third-party codecs and for new media formats such as H.264, Advanced Audio Coding and mp4. Furthermore, the media element in Silverlight has a lot of properties, methods and events that allows a lot of control over the media. Some of the properties are `AutoPlay`, `CurrentState`, `CanSeek`, `IsMuted`, `Position`, `NaturalDuration`, `NaturalVideoHeight` and `Volume`. The methods of the `MediaElement` control include `Pause`, `Play`, `SetSource` and `Stop`. Some of the events are `CurrentStateChanged`, `MediaEnded`, `MediaOpened`, `MediaFailed`, the later allowing some more error handling in case the media cannot be found or in the case the format is incorrect.

This control allows us to bring a symphonic soul to this project. So another aspect of the Bavardica project is the audio streaming, that is now in its most basic form with an mp3 music file playing in the background during the conversation. There are also sound effects that serve as alerts informing that a new user just logged in or that someone just left a new message.

We will now proceed and [describe how the application has been build.](#)

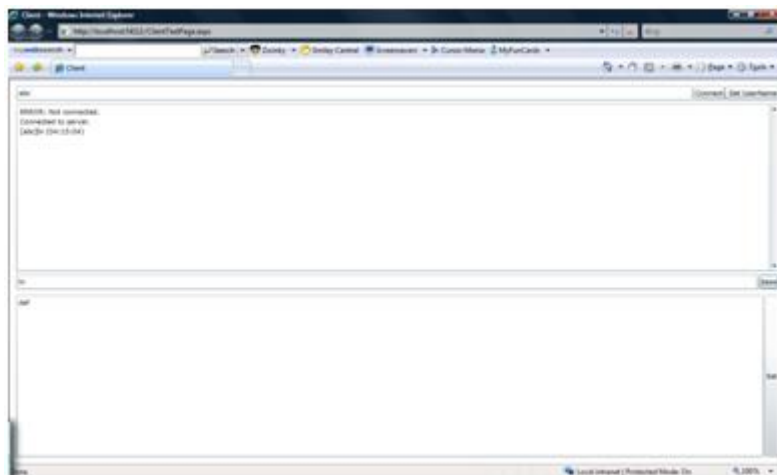
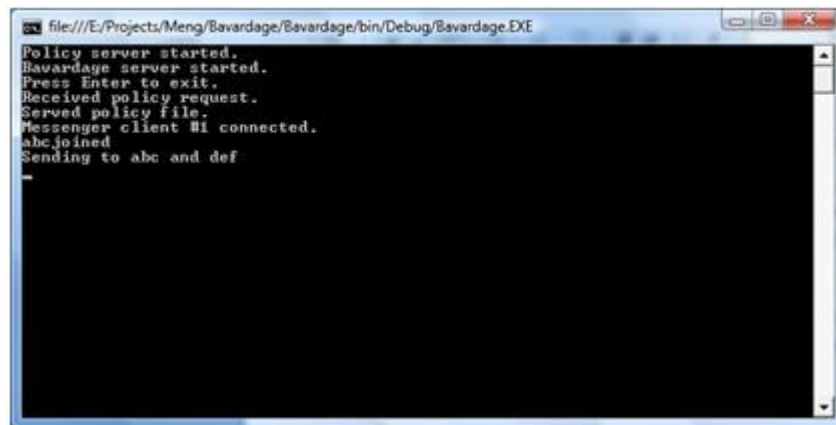
Inside Bavardica - Part 2 : Progress and Achievement of the Project

By [Necemon](#), 2010

As we discussed the [subjects areas](#) related to [Bavardica](#), we can now go through the steps in the actual achievement of the project.

1. Sending a public message over the TCP network from a Silverlight client

The model selected for this project is the prototyping model. Starting first with a throwaway prototype, I could first send simple TCP packets using Silverlight and a console application as a server. It was a very basic chat that let a user, say A send a message to a user B so that no one else will receive the message apart from A and B. The server is a .Net console application and the client is a Silverlight application hosted in an ASP.NET page.



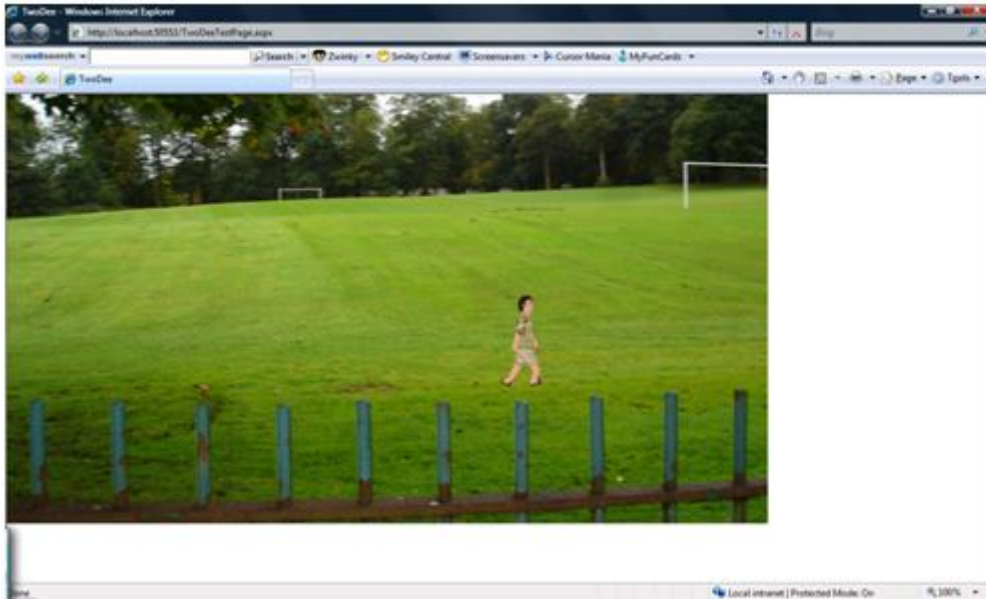
2. First contacts with Silverlight animation

The second program I wrote was a simple Silverlight application with an animated 2d character walking around a park that was actually the background of the web page. This experiment had for sole objective the study of the motion of a character over a 2D scene.

The first key point of this prototype is the way the avatar is animated when a direction key is pressed:

The second key point is that the model used a sprite sheet or frame by frame animation. It consists of sliding horizontally a sequence of adjacent frames with a transparent background, one frame at a time when a directional key is hit.

This approach used `KeyAnimationUsingKeyFrames` but will be dropped later in the project as it involved no interpolation and that it would have required a lot of time drawing each character in every single position.



3. Making the server lies silently in the background (As a web service)

It was planned to integrate the two prototypes so that users could both chat and move around the 2D scene. It was also planned that the server application lies silently in a web service so that a console application need not to be kept open throughout the operation. Both goals have been reached in the next prototype.

The fact is that I thought about building a more robust server application based on which I would build an evolutionary prototype that will gradually grow with the project. That server had two main requirements:

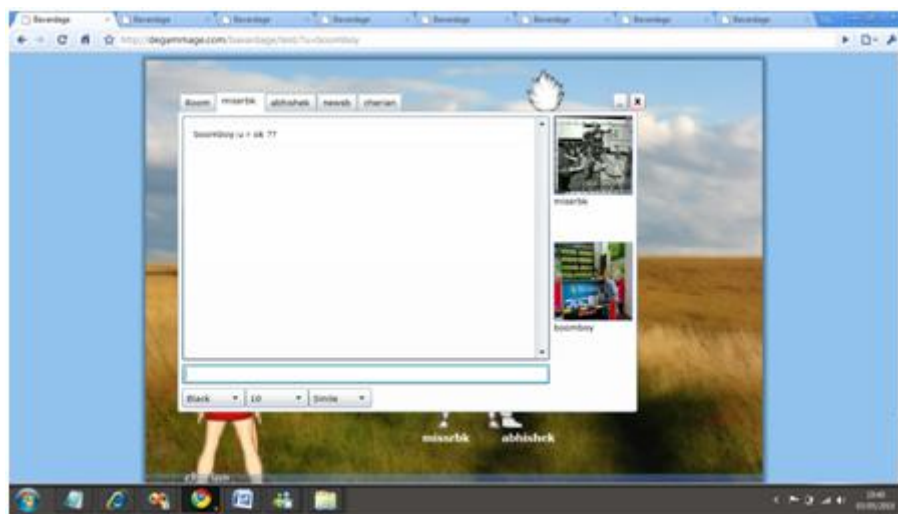
- It had to get rid of the console application. It should just be a silent process in the background. For this purpose, I just used web services.
- The server had to push data to the clients. That is sending data even when they are not requesting. In the classic HTTP model, first the client makes a request to the server and then only the server can respond. A solution to this issue was to use the WCF (Windows Communication Foundation) Duplex web services; and that's what I did. That's an asynchronous two ways communication model.

It's worth observing how the [WCF Duplex web services work](#)

4. Focus on One-to-One Chat

While the public chat happens in a common window, one-to-one chat involves an individual window for each conversation. The organisation of multiple conversations in Bavardica comes down to tabs, using the Silverlight tab.

The Room tab is the one used for public conversations. To start a private conversation, a user just need to click on the name of the person he wants to talk to (There is a list of the available usernames right in the public room).



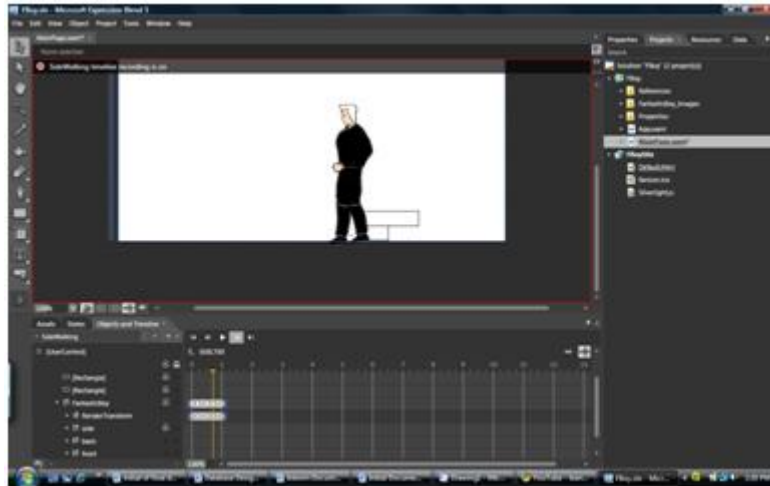
5. Drawing and animating characters

The next step was the design and animation of 2D avatars. There was already one avatar in the previous prototypes but it is not fun if everyone looks the same in a virtual world. Further, that particular avatar had been borrowed from some other Silverlight developer (Darren Mart). The previous model presented a lot of disadvantages. Besides the fact that a sprite sheet animation doesn't provide interpolation between frames, that model also involves a lot of drawing. Each individual position should be designed. But the biggest issue in our context is the nonexistence of customisation in that model: one of the first non-negotiable goals in this project was the ability to mix and match clothes and other outfits as well as their colours. This task is quite simple in a 3D world. It is easier to just swap meshes and textures but in a 2D world, it's a bit tricky to change a part of the outfit, unless we choose to draw a sheet for every conceivable combination of outfit pieces. The solution implemented in Bavardica was rather to create a set of baseline images that would collectively form a character. Therefore I ended up drawing my own characters.

So even if this is mainly a programming project, and not a graphic project, it's worth mentioning that quite a lot of time and effort goes in drawing and animating original characters. Customization is also an important feature of a virtual world. Users should be able to choose how they look and how they are dressed. For this project, I chose to draw four basic characters (2 males and 2 females) and at least two costumes and hair styles for each character. Due to the time constraint again, I could not draw and animate more than that. The operation involves fully drawing and painting the bodies then drawing clothing over. Three basic skin colours are available and the users can change the colours of their clothes and hair. Here is a preview of one of the character:



The characters are basically animated to walk, to sit and to stand. The problem with animations is that it had to be done every time from the three angles (front, back, side). There is also an animation to make them look alive even when they are standing idle. Those models used the `DoubleAnimationUsingKeyFrames` model by opposition to the previous one who used `KeyAnimationUsingKeyFrames`. Microsoft Expression Blend had been used to process the animations:



There are two main advantages to the animation system that had been used:

- Silverlight's interpolation makes things silky smooth
- The characters are customisable. They can just shift boots or hair style (or anything else) as the users want.

Unfortunately, handling characters in a virtual world is not only about design. It involves a lot of coding. Let's start with the creation of a character.

A few array lists are maintained, offering a variety of outfits to the user, as well as shapes and genders

According to the choice of the user, the relevant parts are exposed and the parts that are not relevant are hidden. However, the changes in colour follow another logic.

What happens here is that the source file for each part of the body is fetched from a different directory that changed according to the choice of the user. Later in the application, when the user can change colours about all the parts of their outfits, the same logic is applied.

This was about the character creation. Once the character is created, details about its outlook are stored in the database as expressed in the database design. From there, every time the user comes online, the server can send all its details to all the other clients. As soon as a client is aware of another user online, he has to represent the relevant character in his scene according to the data that had been submitted:

Setting the other characters details work as in the character creation. It's mainly about hiding and showing images.

Now that we have seen how the code involved in characters appearance works, we will now attack the code involved in animation and motion. It starts with a key pressed by a user that is handled by Silverlight. In our example, we will consider the right key:



6. The Cybavard

After the character design and animation, an emphasis has been given to the Cybavard, the artificial intelligence part of the project. The logic behind the Cybavard is inspired from the A.L.I.C.E bot introduced earlier in this presentation. What is interesting about A.L.I.C.E is that it was based on an XML dialect called AIML (Artificial Intelligence Markup Language). Due to the universal nature of XML, it can be used with a wide variety of language. The point is that, as this is a .Net project, AIML could also fit to the Cybavards. It was all about finding a C# AIML interpreter. For this project, I used a .net library file, AIMLBot.dll (by Nicholas Toll), released under the GNU General Public License. That program offers a lot of advantages:

Very small size: about 52 k

Very fast: can process 30 000 categories in under one second

Means of saving the bot's brain as a binary file

A simple and logical API

The inner working of this program is also fairly simple. It's based on pattern matching: for each sentence entered, it checks in its knowledge base which stimulus response would fit best:

So what I am doing is adding a reference to the binary file from my web service, I also upload the knowledge base on the server. The knowledge based is mainly a set of AIML files that I can edit as per my requirements. I could also add my own AIML files. From the cybavard class, what I had to do was creating a bot object and loading the AIML files.

In the following part, we will go through [the design of the characters](#).

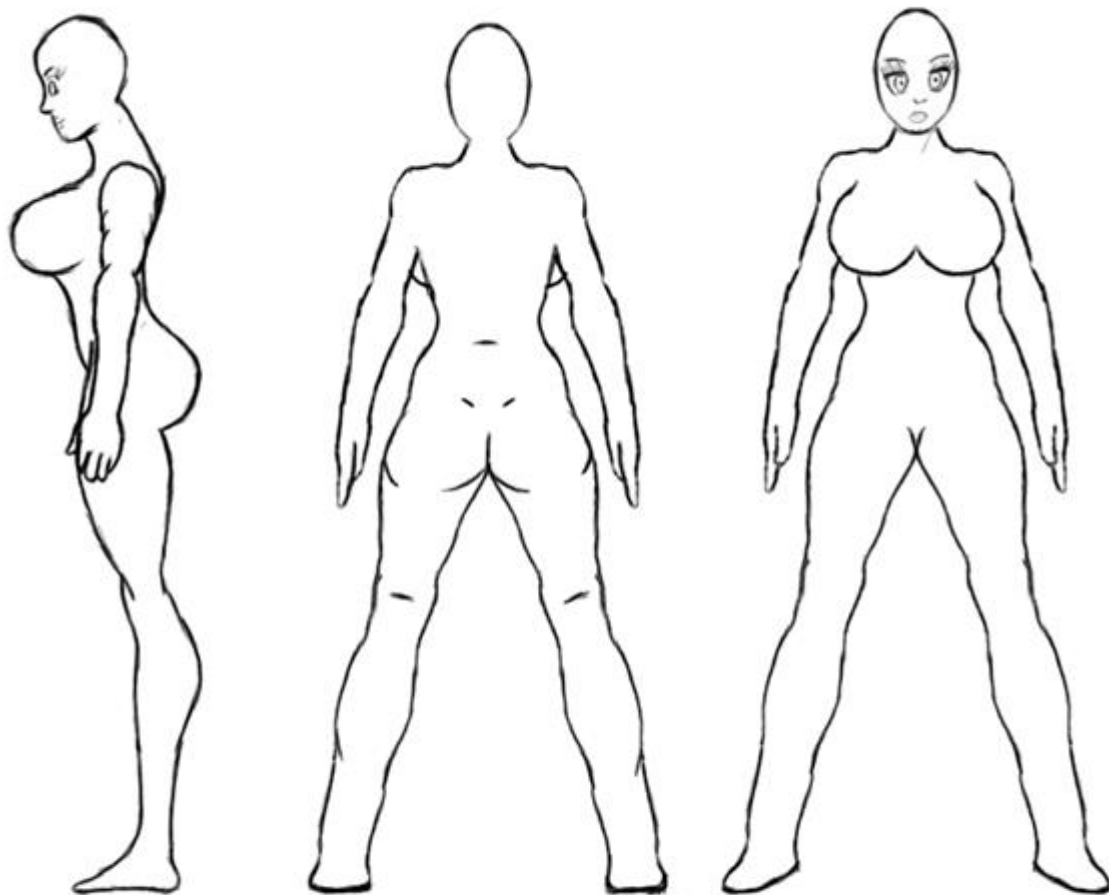
Inside Bavardica - Part 3 : Design of the 2D characters

By [Necemon](#), 2010

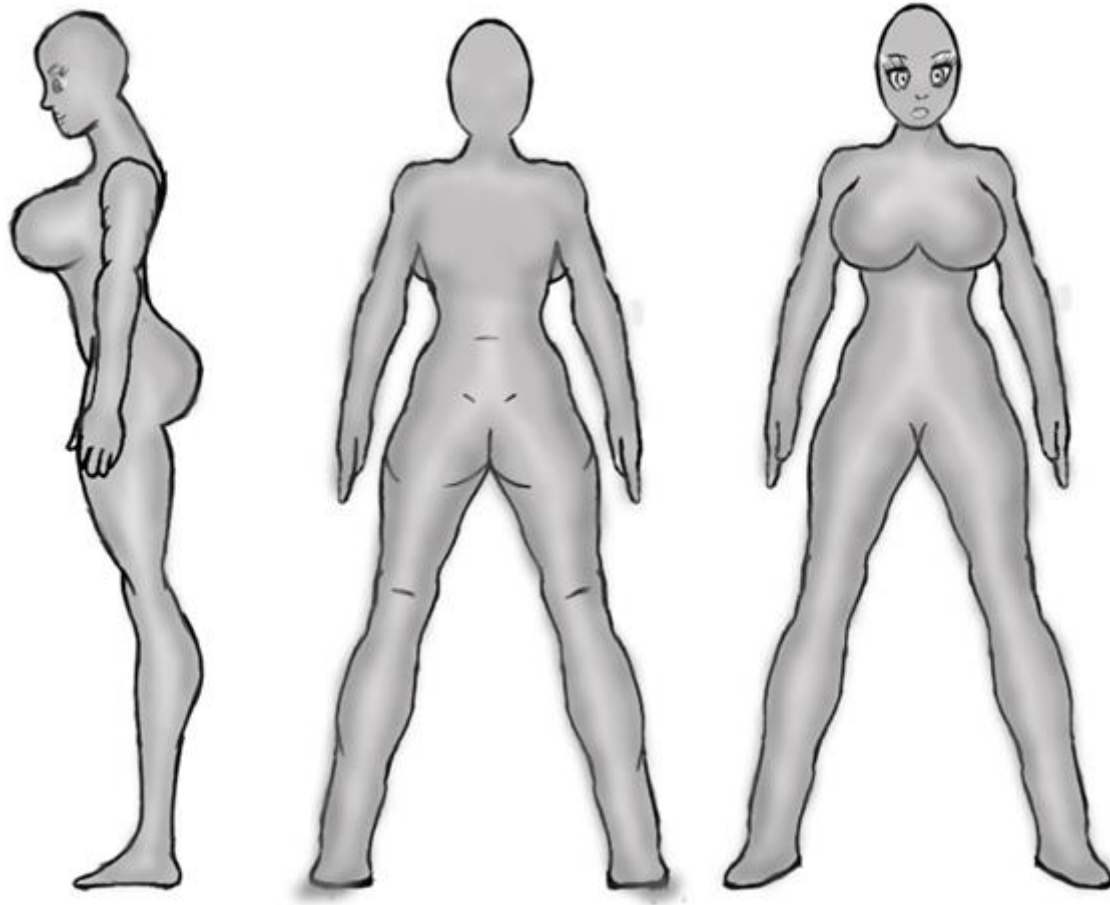
How I draw the 2D characters

Due to the nature of the drawing art, which is not really computer science related, I thought it was best to describe this process in another part right after [the progress description](#). This work had been a full part of the project and even though it does not directly concern programming (before we get into animation), it is also a computer work involving a pen tablet and some image processing software (Adobe Photoshop). The following screenshots describe the steps in the drawing of one character (out of four).

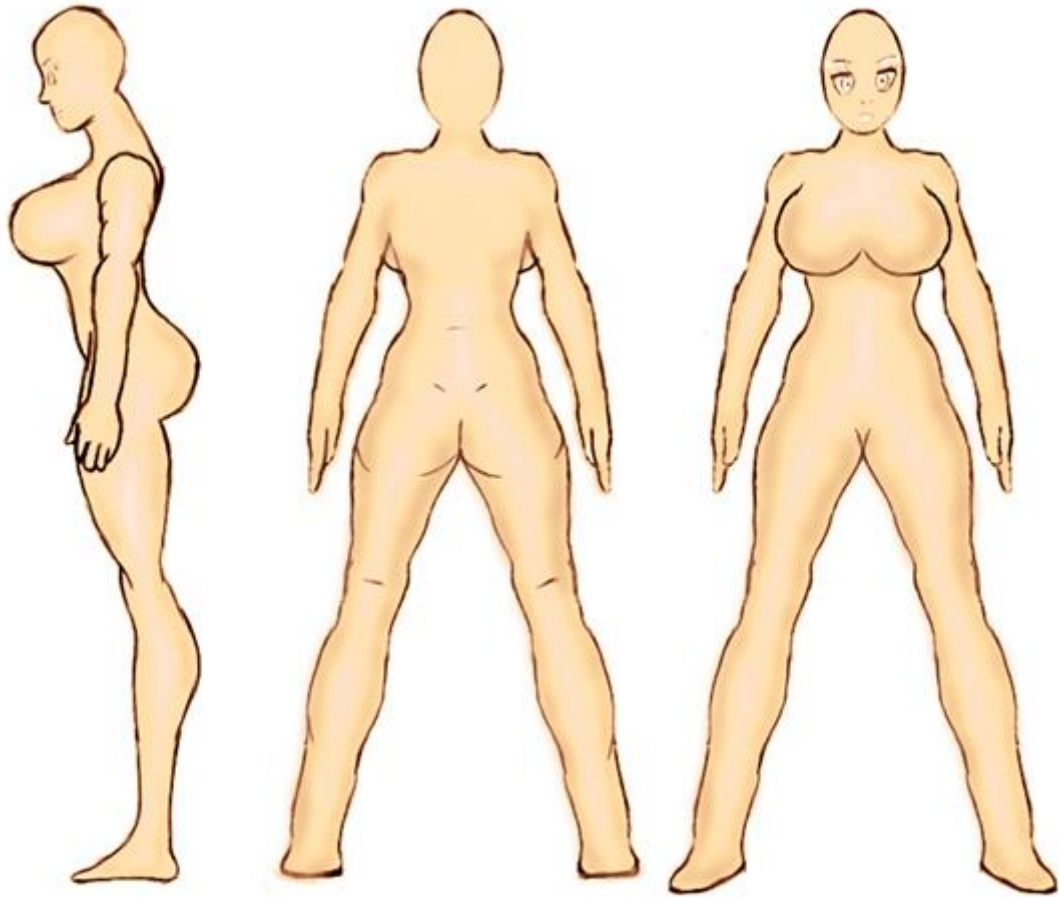
A. Sketching



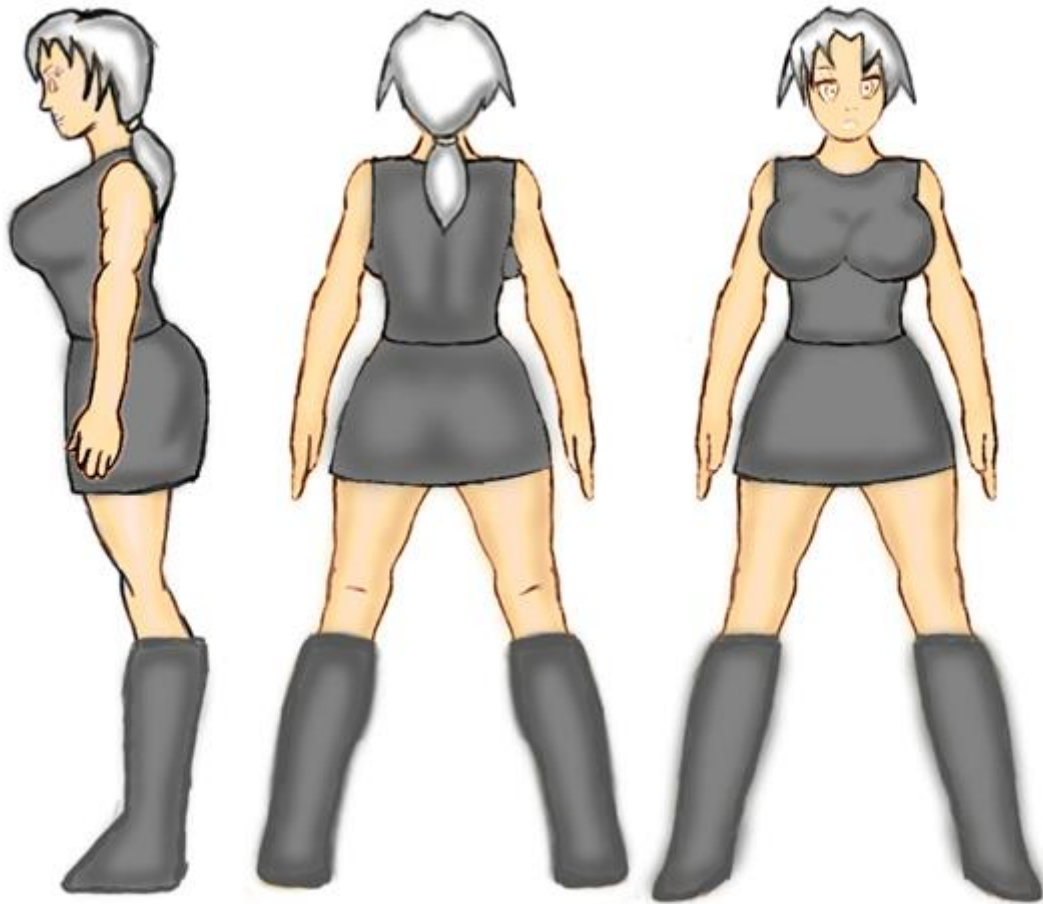
B. Filling



C. Coloring



D. Clothing



For details about animation techniques, you can check [those tutorials that Darren Mart wrote](#). To check out what's coming next with Bavardica, [move on to the next part](#).

Inside Bavardica - Part 4 : Scope for future improvements

By [Necemon](#), 2010

This is the last part of the [series](#) on [Bavardica](#). Apart from a user friendly and colourful user interface, the other features include the edition of the [character outlook](#) in real time (while on the scene). Here again, hair style, skin, clothes and shoes can be changed. Additionally, colours of all those garments can also be changed. The way it works is pretty similar to the way a character is created.

In the meanwhile, some other functions had been added:

Collision detection: This is to make sure that bavards don't get out of the scene boundary. Also, they don't walk inside each other when move vertically. This feature is based on simple "if conditions" that check whether the move doesn't not lead outside the scene or at a position occupied by another character.

Bubbling: That function is used to process text entered in bubble (graphic mode). Every time a user sends a message, it appears on the scene as a flying bubble that gradually disappears. An animation had been designed for this purpose. So both motion and opacity animations are applied to the bubble object. Each key pressed is processed individually as the bubble does not rely on a textbox. The main issue in this task was that the positions of special characters change according to the keyboard type. Finally the bubble handles pretty smoothly conversations from American and british keyboards.



More rooms

The current application contains a single scene where all the public conversations happen. A good extension could be to have many scenes and a map to navigate among the scene.

More music

The music background is always the same mp3 track playing all over again. It is intended in the next version to have a playlist of tracks as music background. Further, since there will be many rooms, each room will have a different theme and a different kind of music.

More cybavards

Due to the large amount of time and data required to simulate the talk of a single cybavard, it has not been possible to provide more than one of those cyber bavards. The process in their creation is pretty simple. It's mainly a matter of time. For instance, the cybavard in this demo has a knowledge base of nearly 4 megabytes (of text).

More animations for a better interaction

The only animations available at the moment are walking in all 4 directions and breathing while the character is idle. It is planned to add more animations like dancing, running, jumping. There will be some moves to express the character's mood. At a later stage, there could be animations that involve more than one character, like hug or hand shake. It is also planned to have more animations for characters to express their moods.

A working version is available at <http://bavardica.com>

10 things I learned from building Bavardica

By [Necemon](#), 2010

This is just a summary of the most important lessons I learned from building and publishing Bavardica. I sincerely hope you get some useful information (or reminder if you already knew all of this) from that. (PS: a working version of bavardica is available at <http://bavardica.com>)

1. Nobody will use an application unless it provides some added value, something interesting they don't already find in what they normally use. This is so true, even for free applications. They might try it but they would give it up quite soon if there is no actual incentive.

2. The very fact that they are asked to register will discourage many people from trying a web application. The most obvious causes might be that they don't want to waste time doing it, or they don't want to give away their details for privacy purpose. So at least a trial version should be offered to visitors without registration so that they can see what it's all about before they make any decision.

3. The various commands and features are (only) obvious to the developer who built the application. However, they may appear as strange and complicated to use for some users. The application should therefore be as simple as possible and there should be some hints for every command (right inside the application, not on an external file).

4. Google is your friend (well, I guess search engines in general). I spent a lot of time reading about Silverlight at the beginning of the project but whenever I had a doubt, I relied more on web searches than on books. I could find most of the answers I was looking for in technical forums. I could also find related work from senior developers (mostly [Anoop Madhusudanan](#), [Tomasz Janczuk](#) and [Darren Mart](#) in this particular occasion).

5. Client-server architecture and silent web services. This one is purely technical. I learned how to make an application lies silently in the background as a web service or a windows service; And I learned how to push data to the client over HTTP using a WCF Duplex Polling service.

6. The more choice, the merrier. Users enjoy having a lot of content to choose from. They like to express their personality and. to customize their character. Art and media requirement consume much more time than actual coding, but isn't it worth it? In the coming versions I would definitely be adding more rooms, more clothes and more animations. But...

7. **Not all at once.** It's important to move one step at the time not to get overwhelmed. It's all about priority management.

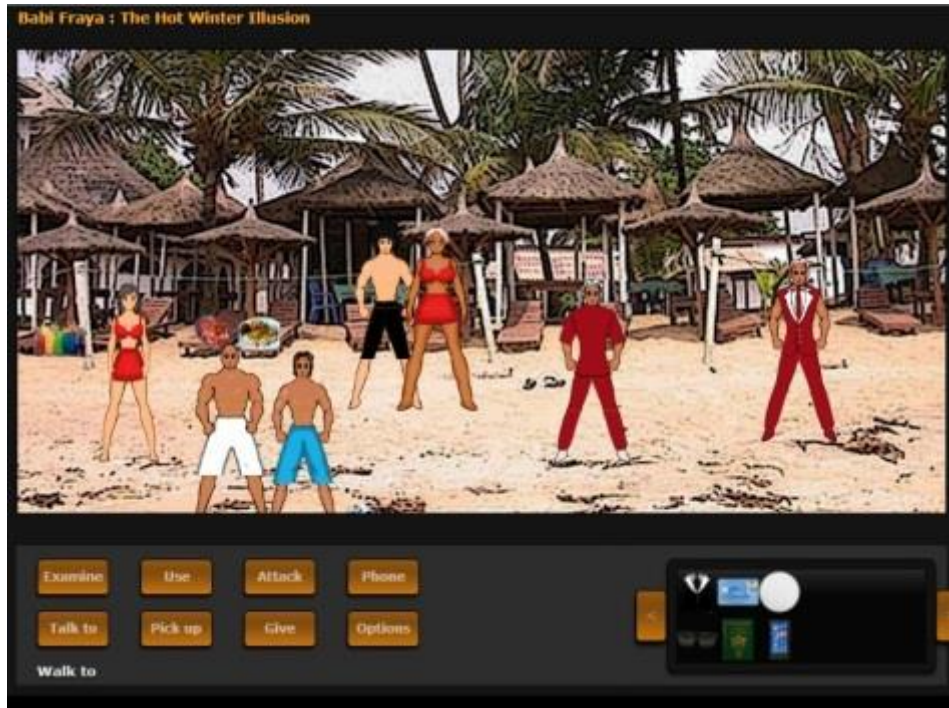
8. **Work results can be rewarding.** I am mostly talking here about the satisfaction and sense of accomplishment one's gets from work. It feels good to accomplish something, even if it's just a small thing for now. It motivates me to keep going.

9. **Feedback is priceless.** I could not learn all those lessons I am describing in these posts if I didn't have some guys and girls to show me my mistakes, to tell me what they didn't like about my work and how I could improve it. I would like to take this chance to thanks all those who tried Bavardica. Thank you for your feedback.

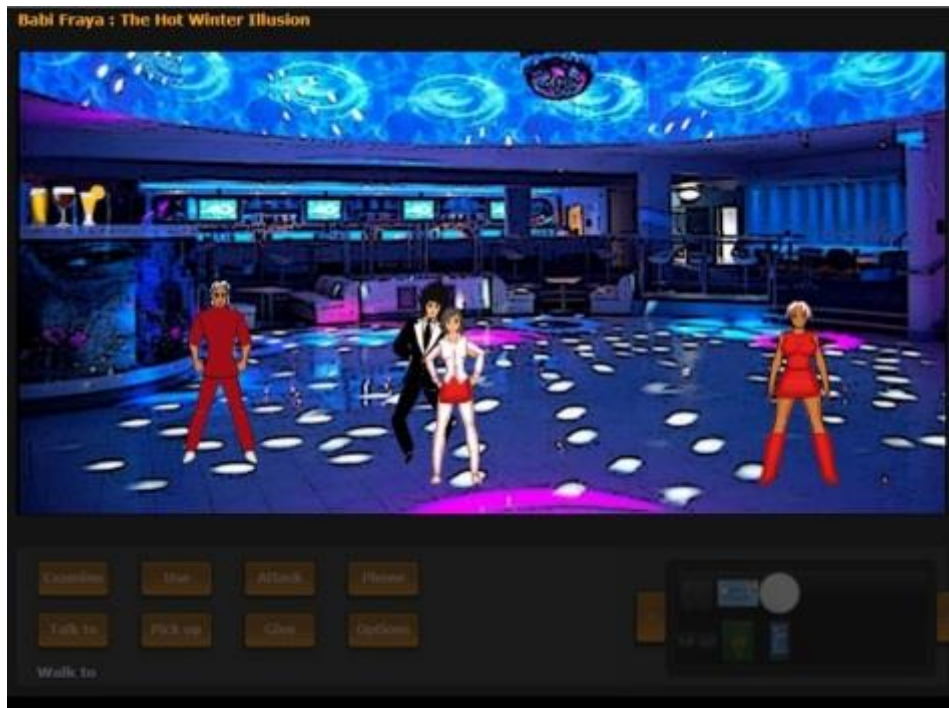
10. **There is a long way ahead** (to build a decent virtual world).

Babi Fraya: The Hot Winter Illusion

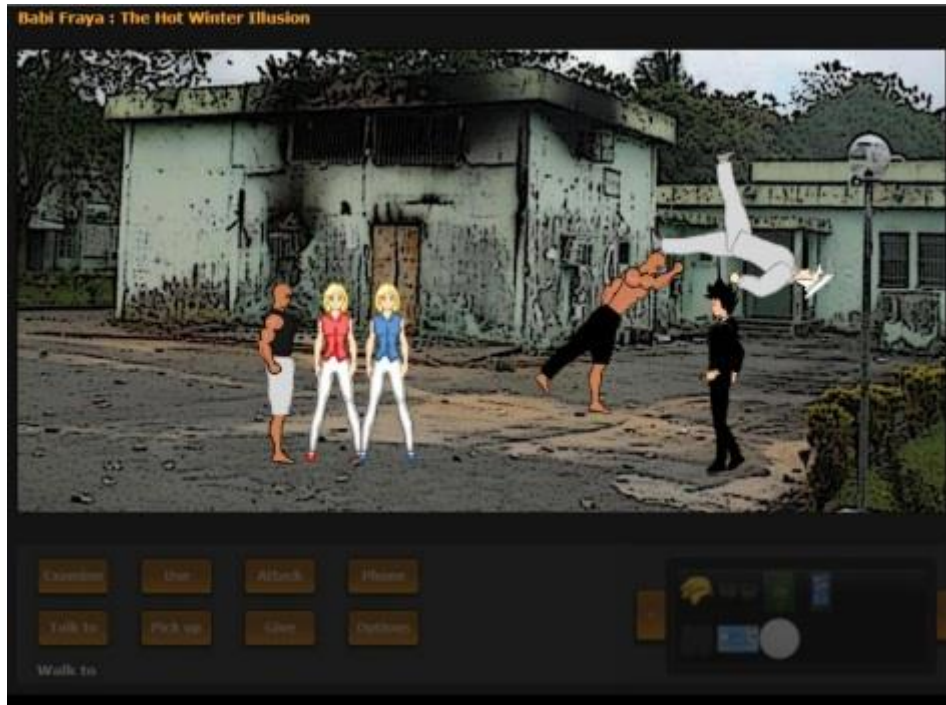
By [Necemon](#), 2013



I just published a beta version of my [first complete game](#). If you have a moment to try it, I hope you will enjoy it : <http://babifraya.com/>



Babi Fraya™ is a ["Point & Click" adventure game](#), therefore a game in which the player assumes the role of protagonist in an interactive story driven by exploration, dialogs and puzzle-solving. It's also a fiction which tells a story, even though the player can affect the story, as there are different screenplay branches.



Scenario

Winter 2010. Just another random New York student heading back to Babi where he is originally from. Great time ahead for Lewis ! Partying all night in Rue Princesse (Princess Street), enjoying the garba on the sunny beaches, hanging out with old buddies and new encounters... Until a sudden change of atmosphere. What could have possibly gone wrong? Nobody knows. It all went really fast... Within a few minutes everything went from harmony to drama: screaming, bombing, looting, violent clashes, food shortage, closure of borders. When Babi Joy becomes Babi Frayeur, what was meant to be a dream holiday increasing looks like a commando mission. What to do now? Fight or Fraya? Fight to Fraya? [You choose](#).



In short, I just have 3 things to say about it:

- This is a [point-and-click adventure game](#), built on [Microsoft Silverlight](#).
- The story offers an off-the-wall perspective of the [#CIV2010 Babi Battle](#), turning dream holidays into a commando mission.
- The game is quite easy and a few parts could potentially be improved but it works fairly well...

Babifraya.com. Any feedback is welcome,

N.

Relaunch : Kpakpatoya 2.0 as an Interactive Platform

By [Necemon](#)

#Kpakpatoya

1. Introduction

[Kpakpatoya™](#) is a repository for what's new, popular and relevant on the Internet with a focus on Africa. Users like you can submit content and decide, through voting, what's good (kiffs) and what's junk (zaps). Links that receive community approval raise towards the top, so the front page is constantly in motion and (hopefully) filled with fresh, interesting links. Kpakpatoya™ is part of the [Evasium Network](#).

"Kpakpatoya" is a term from the Ivorian slang which indicates the act of spreading rumors, passing on supposed stories relating to surrounding things, events and people. In our context, it would therefore be appropriate to understand "kpakpatoya" with regard to scoops, gossips and even compelling news.



2. Historical Context

April 2011. The Ivory Coast was then emerging from a major crisis, a crisis in which information and communication technologies have played an important role in providing practical assistance to related casualties. Beyond simple commenting, hashtags such as #CIV2010 and #CIVSocial were used to spread critical news and provide concrete solutions to these victims. Through donations, call centers and notably Twitter, which then comes up as the ideal tool for broadcasting urgent information in real time.

The Ivorian community on Twitter was growing and booming, with more and more registrations, tweets and contributions. But that community was still relatively small in terms of numbers. For such a small group on Twitter, how to get together, and exchange around common themes? Through the #Hashtags again, of course... On online social networks, the Hashtag is a word or set of words preceded by the symbol '#'. The hashtag is used to centralize messages around a specific term. It acts as a keyword so that users can follow, comment on a conversation or get together around a topic, for example the measures of refueling in times of crisis or even medical emergencies.

But in this post-war period, it was time to switch to less sinister tags. [#Kpakpatoya](#) was quite timely for this small community seeking to gather in an atmosphere that could be described as "cool". It was the tag that these Ivorian "influencers" and "pioneers" on Twitter used to do without local news, inside jokes, viral content, or other kinds of interesting information, including sports events, (technology) conferences, or day-to-day life.

Detecting a chance to improve some socio-technological conditions, I launched the [kpakpatoya.com](#) website in that same period. At the beginning, it was just [a flow of tweets around the #Kpakpatoya feed + a few extra features](#), including a summary of local and international news published in real time. Then I tried with varying degrees of success to offer some applications such as some discussion forums, the blog and the ability to tweet directly from the web site. I had also opened the Twitter account [@kpakpatoya](#) and arranged the mobile application implementation, which still present the news, as well as the Facebook page and the website.

Through this form of ecosystem, and a combination of technical and social skills, the hashtag #Kpakpatoya grew in popularity. By following the thread #kpakpatoya, it was easy to find CIV culture sympathizers, their news, as well as the most popular local themes, even for a newcomer.



3 years later, the Ivorian Web scene turned into a completely different form...

On the one hand, this virtual community has expanded a lot, which is most welcomed, since there is a lot more content, web projects, web entrepreneurs, bloggers, users, etc. Except that a simple hashtag is not enough anymore for an effective exploration of the content. It would not be practical to communicate around a single hashtag. Of course, there is always the option to "break down", to use the alternative hashtags that have emerged in the meantime (although #Kpakpatoya remains one of the most popular tags). That's a good solution, provided one can constantly be aware of new users and tags to follow, so as not to get lost in all this volume of content, and not to miss the most relevant and recent information..

On the other hand, there has been very little evolution when it comes to the #Kpakpatoya project, whether it be in terms of technology, computer graphics, communication, infrastructure, etc. The Facebook page has less than 2000 fans, the Twitter page has around 8000 followers, the mobile application is obsolete, and until recently, the components of the website that had a lot of success in the first few months have been almost neglected (I will spare you the numbers related to the decrease in web traffic). There are almost no contributors to the progress of the project, there are just the users. This could have been a decent community project, but instead it was reduced to a latent state.

It is true that the community was formed in a rather organic way, there was no concrete strategy, everything happened gradually and naturally, which perhaps explains the cool-down after the buzz effect had passed, each party being concerned with their own projects. I must admit that the applications I wrote as part of this project were essentially in maintenance mode and that I had moved on to other things myself. Anyway, no one stepped up.

A wise man once said: If there is anything potentially interesting and you do not understand why nobody does it, it's because you haven't done it either. That's DIY: Do It Yourself.

I therefore take over the lead on this project to try and produce something useful. In recent weeks, my team and myself have been doing some renovations, trying to produce a more interactive platform (with some added value, if at all possible) in order to facilitate meetings, exchanges and content management within the community.



3. Description

Kpakpatoya™ is powered by [Evasium®](#), a small team of technology enthusiasts. We basically make Internet software. We focus on building edutainment-oriented web/client applications.

On Kpakpatoya™, you can discuss pretty much anything, but the most welcomed topics are : Buzz, Clash, Economics & Politics, Culture, Insolite, Local News, Africa, Sport, Diaspora.

Beyond the option to read news, it is now a matter of adding a touch of interactivity, where everyone can share links, discuss, comment and vote on the existing notes. A Kiff is a positive vote (like a "like", a "+1") for a note. A Zap is the opposite, it is a negative vote for a note. Any Kpakpatoya user may kiff or zap a note, and since the score of the note depends on the difference between his kiffs and his zaps, each vote helps to establish the position of the note in the Kpakpatoya ranking. The notes could be listed in order of popularity, to allow the next visitors to spot the best content.

Nothing too complex, it's just a sharing system as found everywhere on the Web. For example, tech entrepreneurs have [Hacker News](#), Americans came up with [Reddit](#), French cybernauts have [tapemoui.com](#). The concept has existed for a while, it's just that there was no Ivorian/African version.

Kpakpatoya™ is available in 2 languages.

In English:

<https://kpakpatoya.com/?lang=en>
<https://www.facebook.com/KpakpatoyaNews>
<https://twitter.com/kpakpatoyanews>

In French:

<https://kpakpatoya.com/?lang=fr>
<https://www.facebook.com/Kpakpatoya>
<https://twitter.com/kpakpatoya>

Kpakpatoya™ may (or may not) become a reference point in terms of news and "affaires" for Ivorian web users. But that's not the main goal here. The current goal is to provide a useful tool for expanding the sharing options for local content, and to implement the long overdue Kpakpatoya renovation.

On a more personal note, I think we should rather see this project as an experience that could potentially help to optimize the visibility of online African content.

For having managed the website, the Facebook account and the Twitter account of Kpakpatoya™ over the last 3 years, it seems to me that many followers expect a better interaction system and they strive for a better visibility for their content, namely their projects, links published, or simple opinions and comments. Therefore, it is possible that this version of the site meets their expectations to some extent.

There are still many things we could add or improve, we can do it as we go along, while taking users feedback into consideration, but the prototype of this new version is relatively stable, and what is needed now is trying it. That is to say, create your account, test the different forms of interaction, provide your feedback and share the link with your contacts so that they can do the same.



Flux Project : Social News Web Applications

By [Necemon](#)



This is just a quick update on my [ongoing side projects](#).

I have been working on a series of [social news web applications](#) where users can submit content and decide, through commenting and voting, what's good (kiffs) and what's junk (zaps). Links that receive community approval raise towards the top, so the first pages are constantly in motion and (hopefully) filled with fresh, interesting links.

Nothing impressive, but it gives me a good basis for making a robust web platform, as in, the common features:

- Security (cryptography, membership, login, registration, password reminder, etc.)
- Database layer / caching (or how to manage an ever growing volume of "big" data)
- Back end services (email senders, pings, http requests, logging)
- A stable [ASP.NET MVC](#) application
- User actions/management : voting, commenting, publishing, moderation
- Assets management (images, javascript, css, minification/bundling)
- Bonus: I am learning a lot when it comes to deployment (IIS settings, virtual directories, ports, application pools, etc.)

In short, the kind of things that could be useful in a modern [web application](#). Actually, this is my most advanced web application so far.

On [Extreme Programming](#) mode

I am still interested in building [virtual worlds](#) and [online games](#), but I thought I would release something (anything) at regular intervals. That way, I can improve as I go, learning/shipping at every step, instead of working on one thing for many years and come up with something that's not really practical/useful. So, that's the idea : getting stuff out there early, as soon as I've got something that covers the basics, and then build it up. If my experience with web projects has taught me anything, it's that people who use apps will suggest or ask for things that the developers wouldn't have thought of doing for themselves, and some of them turn out to be really good ideas even if we don't always quite see the potential at the time. Also it helps getting a better idea of priorities.

Anyway, that basic version might already be quite useful in some contexts, as news portal, one of the flavours being [kpakpatoya.com](#). you can think of it as a small bilingual Reddit for African news. It's starting to have a decent audience (with [8000+ followers on Twitter](#) and [2000+ fans on Facebook](#)) and some of the followers may welcome an opportunity to browse more content and to interact around it (share, comment, vote and add their own content).

So that's what I have been up to recently, each site having its own topics and they are all available online (in English and French) :

[flux.evasium.com](#) (Education, Science and Virtual Worlds)

[notiflux.com](#) (an exploration of online content on Personal Development and Winning at life)

kpakpatoya.com (a repository for what's new, popular and relevant on the Internet with a focus on Africa)

degamage.com (essentially geeky entertainment)

appuyage.com (an exploration of psychology, sexology and interpersonal relationships at the emotional level)

So [I did a game](#) the year before, shipping these web portals this year and the coming months sounds promising in terms of technical improvements and new features. Let's see.

Stay up to date with your favorite topic on social platforms

By [Necemon](#)



Having written [Notiflux™](#) and [AUDE](#) (Automatic Updates Distribution Engine) some time last year, I thought I would play around with the [Facebook](#) and [Twitter](#) APIs and keep posting links to some fresh news and articles in real time.

So depending on the kind of things you like, you might want to join or follow some of those pages:

[Degamage™ UK](#) : PC & Web gaming news.

->[Facebook](#) ->[Twitter](#)

[Evasium Park](#) : A [2D Virtual World](#). We are currently working on the version 0.2. In the meanwhile, here are regular news about virtual worlds.

->[Facebook](#) ->[Twitter](#)

[Notiflux™](#) : Your flux of notes and notifications. All about winning : strategy, tactics, philosophy, marketing, business, etc.

->[Facebook](#) ->[Twitter](#)

[Evasium®](#) : Posting news about fun learning, education through technology.

->[Facebook](#) ->[Twitter](#)

[Evasium Developers](#) : The nerdy one. Latest and best technology articles about programming in general and Microsoft C# .NET in particular.

->[Facebook](#) ->[Twitter](#)

Take your pick!

N.

Furtivue (Or How To Send Furtive Messages Like A Ninja)

By [Necemon](#), 2012



Furtivue : a furtive view over your messages. (www.furtivue.com)

Furtivue is a web service that allows you to send temporary (self-destructing) emails, as in, you can control how long the message is going to stay with the recipient.

When they open your Furtivue message, it appears for a specified number of seconds, then self-destructs. The duration will depends upon the length of the message (Or you can choose how long you want the message to be displayed).

How it works:

User A writes a message to user B, a link is sent to B. When B clicks on the link, the link is de-activated (therefore can't be used thereafter), the message is shown from a Silverlight canvas and disappear after the predefined time is over.

Purpose:

This could be used for sending confidential/secret information (such as passwords), or any other data you don't want the receiver to keep permanently in their e-mail box : for some reason, you may want to express yourself without leaving a permanent record.

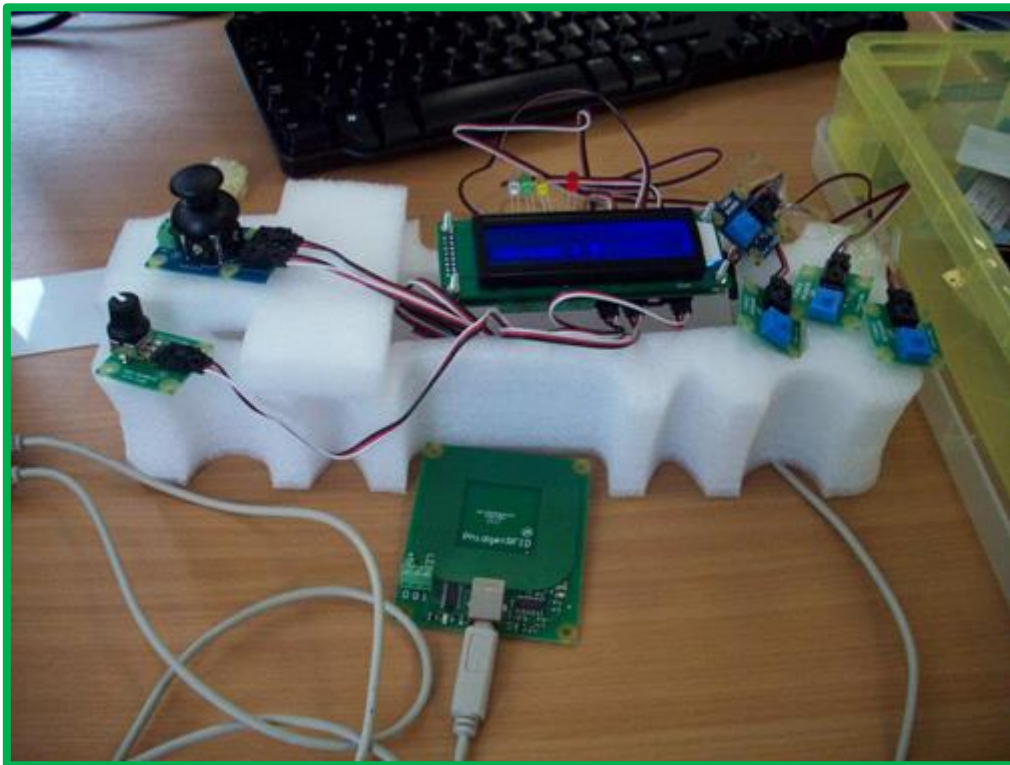
Now what?

You can [try it here](#) or [download/explore the source code from here](#).

Chapter 9

Research and Case Studies

*L'informatica è un'esperienza che ti arricchisce molto
Almeno stando a quello che dice il mio estratto conto !*



Phidgets: First Steps In Robotics?

By [Necemon](#)



Phidgets are a set of "plug and play" building blocks, some low-cost electronic components that you can control from your personal computer via USB. They provide you with some [extra input/output methods beyond the classic](#) mouse + keyboard + screen.

As [Harold Thimbleby](#) mentioned in his book [Press On](#), phidgets are a very nice way to get into hardware programming, as you may want to build real systems, not on screen or web browser simulations : phidgets are so-called because they are the physical equivalent of on-screen widgets (Windows Gadgets).

Phidgets = Physical + Widgets
(Widgets = Windows + Gadgets)

Where to find them ?

there are various official phidgets dealers [spread across the world](#). For example, if you live in the UK, you can check out [robotshop.com/uk](#).

Getting started

All the USB complexity is managed behind the Phidget API (Application Programming Interface). Applications can be developed quickly by programmers using their favorite language: C/C++, C#, Cocoa, Delphi, Flash AS3, Flex AS3, Java, LabVIEW, MATLAB, Max/MSP, MRS, Python, REALBasic, Visual Basic.NET, Visual Basic 6.0, Visual Basic for Applications, Visual Basic Script, Visual C/C++/Borland.NET, etc. (If you can't code you can also use software such as Microsoft Robotics Studio or even Microsoft Excel).

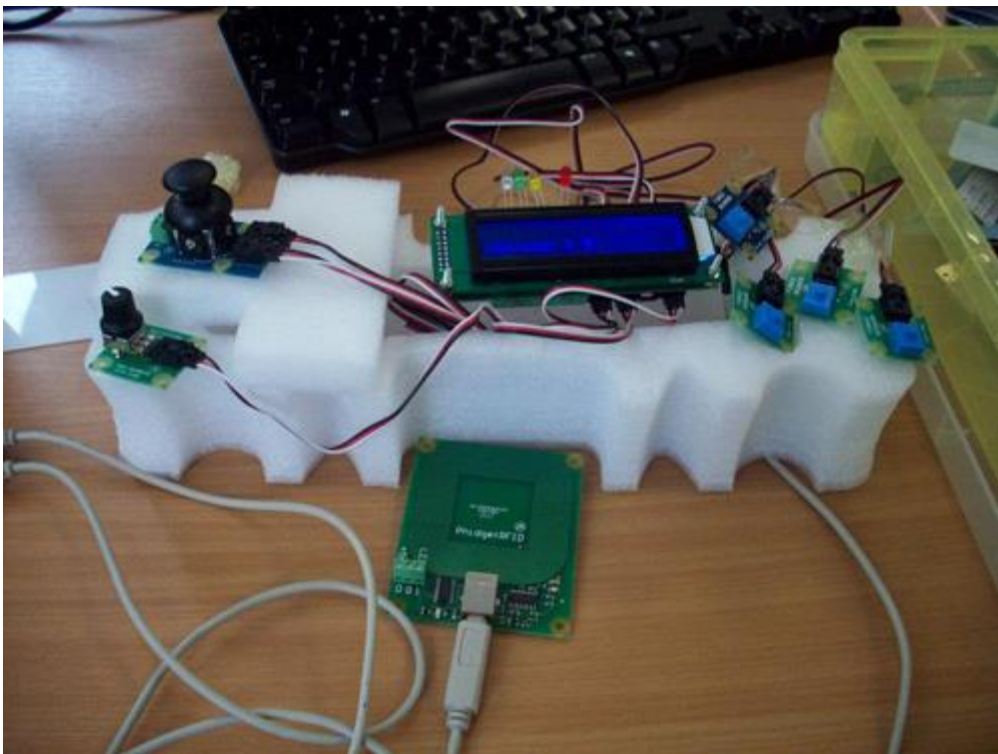
All you need is the drivers that you can get by downloading [the relevant installer](#). Similarly, you can [access the various manuals, the samples and the API sorted by language from here](#)

For example, as a C# developer, just download those and you are ready to go:

1. [Getting Started Guide](#)
2. [Windows Installer](#) (32-bit)
3. [Code Samples](#)

A practical example

Allow me to introduce the Quick-&-Dirty-Pad (while I think of a better name), a "game controller" I built with my partner [Daniel Williams](#), while we were back at [University](#) during the [Interaction Technologies](#) classes led by [Dr Parisa Eslambolchilar](#).



Yes I know, it's not that great, but hey, everyone needs to start somewhere...

We used it primarily to go along with a flight simulation game we programmed, we used force sensors for shooting, a joystick for direction, RFID tags for user authentication and a rotation sensor to regulate the game speed. The game events were reflected through a LCD screen and a few LED indicators.



(Thanks for reading so far. The rest is for the nerds only. This is the part where we briefly discuss the key technical details.)

The game itself was written in C#. It was inspired from a [Riemers XNA tutorial](#). The key parts of the code include the navigation system:

```
if (keys.IsKeyDown(Keys.Right))
    leftRightRot += turningSpeed;

if (keys.IsKeyDown(Keys.Left))
    leftRightRot -= turningSpeed;

float upDownRot = 0;

if (keys.IsKeyDown(Keys.Down))
    upDownRot += turningSpeed;

if (keys.IsKeyDown(Keys.Up))
```

```
upDownRot -= turningSpeed;
```

The aircraft keeps moving forward automatically and the above code is used to control the rotational movements. One of our challenges was to incorporate the phidget code for the joystick into this part of the code such that the joystick behaves as arrow keys.

First of all we created a new input wrapper which is the class that contains the sensor change handler to detect any changes in the movement of the joystick. The input wrapper class implements the interface kit object which detects any input from any phidget

```
InputWrapper iw = new InputWrapper();

ifKit.SensorChange += new SensorChangeEventHandler(ifKit_SensorChange);
```

The first section of the next code controls the x axis of the joystick. The output values of the joystick are between 0 and 999, so for any value greater than 500 the plane would rotate to the right and any value less than 500 the plane would rotate to the left. The second part of the code is used to control the Y axis of the joystick. This time any value greater than 500 would rotate the plane so it's facing upwards while any value less than 500 will rotate the plane so it's facing downwards.

```
if(iw.XJoystick>550)

    leftRightRot = (iw.XJoystick-500)/10000;

if (iw.XJoystick < 450)

    leftRightRot = (iw.XJoystick-500)/ 10000;

if (iw.YJoystick > 550)

    upDownRot = (iw.YJoystick - 500)/10000;

if (iw.YJoystick < 450)

    upDownRot = (iw.YJoystick - 500) / 10000;
```

The LCD screen was used to display the current player and their score. The user must scan their id to start the game or the game wouldn't start. The RFID phidget was used to detect the identity of the current playing user. First of all we created a text LCD wrapper and an RFID wrapper which contain the various event handlers. The RFID detects if a tag is present; if not a message appears on the LCD screen asking the user for their tag

```
TextLCDWrapper tlw = new TextLCDWrapper();

RFIDWrapper rw = new RFIDWrapper();

while (rw.tag == null || rw.tag.Length < 2)

{

    tlw.JustTyped("Tag, Please!");

}
```

When the user scans their tag the RFID will detect if the user is registered to play the game or not and display the relevant message on the LCD screen

```
private void DrawText()
{
    if(rw.tag != null && rw.tag.EndsWith("cb"))
    {
        tlw.JustTyped("Necemon : " + score);

        spriteBatch.DrawString(font, "Necemon - score:" +
score, newVector2(20, 45), Color.White);
    }

    else if (rw.tag != null && rw.tag.EndsWith("8a"))
    {
        tlw.JustTyped("Daniel : " + score);

        spriteBatch.DrawString(font, "Daniel - score:" +
score, newVector2(20, 45), Color.White);
    }

    else if (rw.tag != null)
    {
        spriteBatch.DrawString(font, rw.tag + " - score:" +
score, newVector2(20, 45), Color.White);

        tlw.JustTyped("Unknown : " + score);
    }
}
```

The rotation sensor will be used to control the speed of the plane. The values of the rotation sensor are obtained through the input wrapper class, previously mentioned when describing the joystick. The more you turn the rotation phidget the more the speed increases

```
gameSpeed = iw.RotationValue/100;
```

The following code creates a new bullet every time the space bar is pressed provided the previous bullet was fired more than 100 milliseconds ago

```
if (keys.IsKeyDown(Keys.Space))
{
```

```

double currentTime = gameTime.TotalGameTime.TotalMilliseconds;

if (currentTime - lastBulletTime > 100)
{
    Bullet newBullet = new Bullet();

    newBullet.position = xwingPosition;

    newBullet.rotation = xwingRotation;

    bulletList.Add(newBullet);

    lastBulletTime = currentTime;
}
}

```

Our task was to implement four sensors to control the bullets being fired from the plane. The values of the force sensor are also obtained through the input wrapper class. The following code now creates a new bullet of type 3 every time the force sensors are pressed

```

if (keys.IsKeyDown(Keys.C) || iw.Force3 > 25)
{
    //make the lights blink

    iw.ifKit.outputs[0] = (!iw.ifKit.outputs[0]);

    iw.ifKit.outputs[1] = (!iw.ifKit.outputs[1]);

    iw.ifKit.outputs[2] = (!iw.ifKit.outputs[2]);

    iw.ifKit.outputs[3] = (!iw.ifKit.outputs[3]);

    double currentTime = gameTime.TotalGameTime.TotalMilliseconds;

    //within a certain frequency

    if (currentTime - lastBulletTime > 300)
    {
        //Create the bullet and add it to the bulletlist

        Bullet newBullet = new Bullet();

        newBullet.position = xwingPosition;

        newBullet.rotation = xwingRotation;
    }
}

```



```
bulletList3.Add(newBullet);  
  
//play a sound  
soundEffect1.Play();  
  
lastBulletTime = currentTime;  
  
}  
  
}
```



Paper Prototyping

By [Necemon](#)



Last week, I had to make a [paper prototype](#) for a mobile application I was working on. I ended up building a quite nice thing (picture above) but at the beginning I was not sure about how to go about it. So I just thought I would write about it to share how I proceeded.

The first thing was to do a little bit of research, just to see how people normally do it. The website paperprototyping.com had been quite inspiring for that matter (as well as [that video](#)).

I then headed to the office stationery to grab the necessary stuff. For the whole process I just used a pencil, a pen, a gluestick, some record cards and scissors. That's it.

The next steps were to cut some of the cards to the scale of a (big) mobile phone, wrap one of them around the others and make a hole on each side so that "screens" could go through:



All this may sound kind of obvious but I guess that's the article I wish I could have read when I was starting.

Thanks for reading.

Dual Shock 3 - Part 1: History of game controllers

By [Necemon](#), 2011



This study is about a device which is probably the best joypad of all times: the Dual Shock. It has many variants and we can expect it to keep evolving in the coming years, however, we will focus here on the current version, the Dual Shock 3.




In this article, we will go through the history of the game controller in general and then focus on the history of the Dual Shock itself, so that we understand how it was built. In a next parts, we will carry on with [the context in which the device was built](#) and then [analyse the controller by studying its specifications](#).

Basics




There are many categories of game controllers, among which we will retain the most popular one, the gamepad, as that's exactly what the Dual Shock is. Gamepads, also known as joypads, can have many action buttons combined with one or more omnidirectional control sticks or buttons. They are held using both hands with fingers (typically thumbs) used to provide input; as mentioned on Wikipedia, most modern game controllers are a variation of a standard gamepad. Common additions include shoulder buttons placed along the edges of the pad, joysticks, centrally placed buttons and internal motors that provide haptic feedback (as in, they typically vibrate).

Some history

In his article intituled "History of the game controller", Catalin Ivan stated "Ever since the very beginning of video gaming, the controller has been the best (and usually the only) way of man-machine interaction." That's actually an interesting fact, and it would be good to follow the evolution from that "very beginning" to see how we reached the current gamepad technologies. There had actually been many, many gamepads released throughout the history, we are not going to study all of them, we are just going to review those who appear like important milestones that we observed from a study by John Honnibal and that we are summing up in the following table:

Controller Name	Description	Picture
The TV Tennis Game Paddle	<ul style="list-style-type: none"> - An analogue control based on a monostable - Game-start buttons were located on the console, not on the paddles - One variant was the use a slider control - Some TV games didn't even put the paddle on wires but place them on the front of the game itself - Generally, with the early models, there were no buttons on the pads 	  

<p>The Atari 2600 Joystick</p>	<ul style="list-style-type: none"> - A digital four-way joystick with a single fire button - Again, game-start button on the console itself 	
<p>The Atari 7800 Joystick</p>	<ul style="list-style-type: none"> - A digital four-way joystick with two fire buttons - Simply add an extra-button to the 2600 Joystick 	
<p>The 8-bit Home Computer Joystick</p>	<ul style="list-style-type: none"> - An analogue joystick with a single fire button - Here, the game could be started using the keyboard 	
<p>The Vectrex Joystick</p>	<ul style="list-style-type: none"> - An analogue joystick with FOUR buttons - Games were started from the controller using the normal fire buttons - The fire buttons are labelled '1' to '4' 	
<p>The Nintendo Entertainment System Joypad</p>	<ul style="list-style-type: none"> - A digital joypad with two game-start buttons and two fire buttons - introduced the idea of placing active components and circuitry inside the joypad. - The shape of the game pad, however, is a very basic rectangular box. 	

	<ul style="list-style-type: none"> - The two fire buttons are both red in colour and are labelled 'A' and 'B'. 	
The Super Nintendo Entertainment System Joypad	<ul style="list-style-type: none"> - A digital joypad with two game-start buttons and six fire buttons. - introduced the idea of shoulder buttons - shaped much more naturally to fit the player's hands. 	
The Sony PlayStation I Joypad	<ul style="list-style-type: none"> - A digital joypad with two game-start buttons and eight fire buttons. - adds more shoulder buttons to the SNES pad - an even more ergonomic shape 	
The Sony PlayStation II Joypad	<ul style="list-style-type: none"> - An update to the PlayStation I controller that adds analog controls and a pair of vibration motors. - the analog-to-digital converter (ADC) is in the controller and the interface is digital. - The two motors are fitted with eccentric weights of different sizes, and can be independently controlled by the game. 	

From this brief summary, we can see the different steps in the history of the game controller before the Dual Shock took over. A few deductions we can make are firstly that we slowly went from controllers with zero buttons to controllers with one, two buttons and so on. Those buttons became more and more meaningful with colours and labels. We also observe that the commands slowly moved from the game console to the controller. Further, we note that controllers got more and more ergonomic with time.

Now, let's revisit [the amazing story of the DualShock](#).

Dual Shock 3 - Part 2 : The amazing story of the Dual Shock

By [Necemon](#), 2011



We just reviewed the [evolution of the game controller](#) in general. Now let's dip in the very interesting story of the creation the PlayStation Controller. The facts reported here mainly come from [Kevin Gifford who wrote the "PlayStation 1 Design" article](#) and from [Reiji Asakura who wrote the book "Revolutionaries at Sony – The Making of the Sony PlayStation and the Visionaries Who Conquered the World of Video Games."](#)

To get into the right context, allow me to introduce a few Japanese public figures who are keys in this story. First, Ken Kutaragi, who was chairman of Sony Computer Entertainment. Then we also have the late Norio Ohga (1930-2011) who was president of the Sony Corporation by the time the story happened. And there was, Teiyu Gotoh, the main designer of the PlayStation Project, which is the hero of this story. It's also worth mentioning that the Sony PlayStation was a huge success as it was released. It was actually the largest selling single model the electronic equipment market had ever seen (with 40 million units worldwide without a model change within three years). "One of the main factors that led to the success of Playstation", said Ohga, was the design of the main unit and controller. That's what Gotoh confirmed by stating "I'm convinced that the design contributed to the runaway success of the PlayStation, We had to start the Playstation business with no track record. A major issue was whether software companies would publish titles for it. I thought the first hurdle was to ensure that game creators would like the hardware design. After all, game creators are the machine's first users. We've got it right if they think they want a platform that looks like ours."

For Gotoh, design was the very expression of content. He was aiming for a simple yet original and trendless design. Gotoh was originally a designer of television sets. He was well known for his sophisticated design style and excellent performance on projects like the Sony Profile PRO and the Sony miracle, the VIAO PC. However, the PlayStation project was a completely different matter, as Gotoh himself said: "TVs have a fixed shape and use the same devices. Within those constraints, I struggled to find ways to create innovative designs that are different from other products, how to emphasize the difference. But the game machine was completely different from conventional products, because the PlayStation is a one-and-only product. Although it competes against other platforms in the game-machine market, there are no other PlayStations. Other Sony products categories have their own history and fixed design process, but there were no examples to follow with the PlayStation, only a concept of doing something different from other products in the same genre."

In our study, we are more concerned with the PlayStation controller than the console itself. But the interesting part is that Gotoh designed not only the main parts of the PlayStation but also its peripherals such as the memory card and eventually the controller. However, at the beginning, the controller design didn't know a unanimous approval at Sony. While Ohga approved it, Kutaragi, the technologist, immediately rejected it: "What's this? The shape is original, but it doesn't look very easy to use." It was indeed very different from the conventional controllers which were flat, planar while Gotoh's controller was three-dimensional. He recalls "I have worked at Sony for twenty-one years now, but there has never before been such a difficult product. The controller was considerably harder to design than the console." It took him more than a year to design it. He proceeded by carving a lump of acrylic foam and gripping it repeatedly to see how it felt. The shape of the controller was refined through countless repetitions of this process. A few issues with the flat controllers were that they had to be held tightly during use as the palm was not in contact with the device. This tended to be uncomfortable and stressful. Further, those controllers could not properly fit different types and sizes of hands. That's how Gotoh came with some grips so that the controller could be held as naturally as possible.

At some point, Gotoh conducted an experiment, by gathering some children and make them test his controller. They found it a bit weird at the beginning as they were used to the flat controllers. However, once they learned how to use it, they seemed to really enjoy it. This was encouraging because children are honest in their feedback. If they like something, or if they don't they just say how they actually feel about it. Further, there was another very interesting point in that experiment. [As Harold Thimbleby said](#), children are better testers for devices than adults, as they tend to behave like trolls. They keep trying new things without fearing breaking the device and the fact that they are good at getting rid of any technology preconceptions made them avoid any design assumption. This made them the perfect testers for a revolutionary device like the PlayStation controller, spearheading a new generation of game controllers.

Dual Shock 3 - Part 3: Device Analysis and Specification

By [Necemon](#), 2011

Now that we understood [the context and the motivations](#) that led to the creation of the PlayStation Dual Shock from the first part, we will get to know more about the device itself.



The dimensions of the controller are roughly 6 x 3.5 x 2 inches.

The Dual Shock has on the left side, the same directional pad found on most controllers. However, on the right, the action buttons are labelled with original symbol; this is also an innovation by Gotoh while conventional controllers normally used numbers and letters to label the buttons, as he states himself: "Other game companies at the time assigned alphabet letters or colors to the buttons. We wanted something simple to remember, which is why we went with icons or symbols, and I came up with the triangle-circle-X-square combination immediately afterward." In an interview reported by Games Radar, Gotoh explained how it came up with the symbols "I gave each symbol a meaning and a color. The triangle refers to viewpoint; I had it represent one's head or direction and made it green. Square refers to a piece of paper; I had it represent menus or documents and made it pink. The circle and X represent 'yes' or 'no' decision-making and I made them red and blue respectively."

Further there are also four shoulder buttons on the top, for the middle fingers and two joysticks that improve the experience in some game categories like racing or flight simulation.



The vibration is an interesting effect as the device rumbles when some dramatic events happen during the game. That provides a great game experience.

However, the joysticks and the haptic effects didn't appear in the original version of the PlayStation controller. As a clarification, it's worth mentioning that the Dual Shock is actually an advanced version of that original version.

Now coming to the latest version, which is the one we are mostly concerned with here, the Dual Shock 3, which itself is an advanced version of the Sixaxis, the original game controller for the Sony PlayStation 3. Both Sixaxis and Dual Shock 3 looks the same but they are different in that the Dual Shock 3 is heavier, more opaque and it rumbles.

Even though the Dual Shock 3 can be plugged to the console with a removable USB cable, it's also a wireless device which can work over a Bluetooth connection. An autonomy of 30 hours is ensured by an internal battery. The wireless ranged is believed to top out over 20 meters. (PS3 Fanboys 365)

Ergonomically, all the Dual Shock are pretty much the same. And honestly, they are the most comfortable game controllers I personally had the chance to play with (I had been trying a wide range of controllers over the years). However, one downside of the Dual Shock 3 could be that it is quite lighter than its predecessors. But here again, the player get used to it after some time.

Putting the DualShock 3 in perspective with competitors



Technically, the Xbox 360 provides an interesting option when it comes to batteries. They use AA batteries which can therefore be easily changed. On the opposite, the Dual Shock internal battery can't be changed (Not by the gamer, at least). But when it comes to cables, the Dual Shock 3 uses standard USB cables while the 360 controller requires a specific cable that is unique to that controller. All those factors affect the overall cost of the device from the gamer point of view. However, the Dual Shock 3 is more expensive on the market than the 360 controller (£39 against £29 by the time I purchased mine).

Now when it comes to playability, each controller has its own strengths and weaknesses. Mike Ferro once compared the playability of controllers based on the fighting, racing and shooting categories. If we consider racing for example, The key difference here is in the shoulder buttons that the racers normally use to decelerate. They are much more comfortable on the 360 controller as they show less resistance.

For shooting games, the critical controls are the joysticks. The Dual Shock 3 analogue stick has a slightly smaller dead zone, which offers a greater precision and a better response time to the player. Coming to fighting games, the use of the directional pad is critical to make the character execute precise moves. The Dual Shock 3 is clearly superior in that matter because of the way the touches are partitioned. On the contrary, moves are hard to execute on the 360 controller because the user literally feels they are trying to move a single piece of plastic, as opposed to independent up, down, left and right angles.



Business at the speed of game production: OpenGL or XNA?

By [Necemon](#), 2012



A common start ups question in the gaming industry is: Which graphical environment should we use? As time is money, a crucial factor in a choice is time: not only the time it takes to build complete games with each API but also a fluid timing during the gaming experience to make it enjoyable. In this analysis, we compare the OpenGL industry standard to the Microsoft XNA game framework on the speed factor. The study includes learning curve, programming, modeling and rendering speeds on both sides.

[Download the report here](#)

GOOSE (GOOgle Supply search Engine)

By [Necemon](#)



Google GOOSE is a product I came up with during a marketing exercise. We were asked to think of a new product from a well-established brand that could fulfil customers satisfaction. Please note that the GOOSE product is purely fictional. I don't work at Google and the following product description is not endorsed by Google or any of its employees.

The problem

Shopping is a time consuming process; the rhythm of our lives, jobs and other activities goes so fast that we hardly find time to shop properly. The current technologies provide us solutions to speed up the process by shopping online. But even online, shopping remains a task that involves some effort, thinking and time for every purchase. Sometimes you just don't feel like doing it; or you got a lot of other things to do. Further, we are sometimes so busy that we may just forget to do it.

Another thing is that it may be such a pain. Sure, it's fun to shop for new clothes and shoes. But it's not fun to go get vegetables and meat every week.

The solution

What if you could just decide once what should be there in your fridge at all times and not worry about it again? You could just have what you want at all time without having to regularly spend the effort, time and thinking that food shopping require. And that's what the new Google Supply Search Engine (GOOSE) is all about.

The GOOSE (GOOgle Supply search Engine) is a software platform for a new technology of home fridges which systematically:

- knows what food items you need
- knows when it's missing from your fridge

- check the best offers for those items on the internet
- order the missing food online
- trigger the delivery of those items back to your fridge (actually to your doorstep)

How it works

- On the server side:

The GOOSE consists of a web service where each customer can register and provides a list of the items he needs at all times. The customer can also set special items that will be only delivered under specific circumstances (for example, a birthday cake to be delivered only once at a particular date). All that information is stored in the GOOSE database.

The customer can also set on which frequency he wants his fridge to be filled up, say every Friday afternoon. So, when the time comes, the GOOSE web service orders the missing items that had been communicated from the shop the customer indicates. But an interesting thing is that the customer can give the freedom to the service to choose from which shop to buy. It obviously can't be too far from the customer home but the key is that the GOOSE service can compare the offers from different shops and choose the best offer. The goods are then delivered to the customer's house.

- On the client side:

The GOOSE software works on a new model of home fridge system that contains some special components:

- A touch screen device for the user to set up his options. The screen also let the user see and choose the promotional offers that Google sends him, (by the way, this brings additional revenues since advertisers can pay to show some ads to the users).
- Additionally, all the user inputs can also be done from a regular computer system which is connected to the GooSE account of the said customer. All the changes made will be updated to the GooSE database and communicated back to the fridge and vice-versa.
- The fridge is equipped with some devices that track the fridge content and the checks in and checks out of any food items. Those devices are mainly camera, infrared devices and sensors. Bar codes could also be used to identify complex items.

More details

The software would have to interact with the internet for online purchasing and content recognition to some extent. This means the fridge should have access to a Wi-Fi connection. The insides of the fridge would have to be made in a specific manner for content recognition. For example the egg tray, the compartments for bread, vegetables and fruits need to fit a small sensor to recognize if the item is present or not. For example, a beam of light could be cut when a particular product is present uninterrupted when the product is absent. However this wouldn't completely solve the problem as the software would have to recognize what kind of item is present. For example if we take the case of fruits and vegetables, the software would have to know which fruit or vegetable it is, this could be done by having a camera in your fridge which is also capable of counting the number of fruits or vegetables. A barcode reader could be used for those items that come with a bar code. Barcode readers and cameras can be integrated in their usage. Once the camera recognizes the barcode it is then sent online and checked with online databases. The same technology applies for the barcode reader application on Smartphone's.

Now that we have come to an understanding on how content is recognized, what do we do with this information? Based on this information a shopping list is generated and stored temporarily until next purchase is made. The shopping list can also be categorized based on various sections such as:

- o Most wanted items such as soft drinks or any item the customer would be keen on having at all times.
- o Least important items which are items that are customer or not very particular they should have at all times.
- o Short lived items such as milk and other such products that have a short life span. The software would display the date on which the last purchase was made and would give you the option to add the number of days for which that product would expire.

The software would also be capable of reminding you prior to the day of expiry that the item needs to be consumed.

When the software comes to use for the first time a general shopping list is created regardless of categories. Google having kept every bit of information of past purchases then gradually learns what type of customer you are and offers a shopping list based on auto categorization which you either accept or deny.

Potential customer base:

Goose is a software product that searches and orders food items, based on the purchase history of the items which were previously stored in the fridge. It automatically searches and orders the product on a scheduled date, so people who are working could use it at their convenience. Further GOOSE mainly targets

- The young people, who are the most familiar with the internet technology and online shopping. It would perfectly fit students who move away from their parents and are not used to food shopping.
- The parents, who are responsible for their whole family food supply and sometimes have a hard time dealing with it (especially the single parents).
- The developed countries: places with high internet penetration and widely spread home delivery.
- Additionally, GOOSE could equally assist both genders: men are typically the most uncomfortable with food shopping so it could be a great help to them. Conversely, women who often have all the burden of this task would be relieved in a high extent.

Our product Goose doesn't order the product in the single store, it compares the product in the many stores available in the market and automatically order the cheapest or best product. However, some stores can be promoted by paying some advertisement fee. That would be the second form of profit for GOOSE after the fridge software licensing fees.

Challenges

The first challenge met in the implementation would be for Google to find a partner that would manufactures the fridges that would support the GOOSE System, as Google is not in the hardware industry. It should be fairly feasible to collaborate with a well establish fridge manufacturer (like LG, Samsung, etc.) thanks to the notoriety of Google as a brand.

Another challenge would be the market penetration. A lot of market research need to be done to know how to build a satisfying product and once the product is ready, it will need to be promoted. But promotion is not really a problem, Google is the king of online advertisement after all.

Final Reminder

Before we conclude, I just thought I would remind you that this document contains the Volume 2 of the Album, and the Volume 1 is also available. You may download, (re-)read or share any of the various chapters and volumes independently/separately, depending on your interests. The PDF, EPUB, MOBI/AZW3/KF8 (Amazon Kindle) and MP3 formats are available.

To get them, just click on any of the links you like below or go to TheAlbum.NeceMoon.com (or necemonyai.com/Blog/page/The-Album.aspx)

The NeceMoon Album (complete)

Volume 1: Moon Light (softcore)

[Chapter 1: Strategy and Tactics](#)

[Chapter 2: Digital Marketing and Web Visualisation](#)

[Chapter 3: Corporate Worlds and Emerging Markets](#)

[Chapter 4: Quick Wins, Tricks and Tips](#)

[Chapter 5: Transition - Extra Thoughts and Sharp Fantasy](#)

Volume 2: Full Moon (hardcore)

[Chapter 6: Software Development and Engineering](#)

[Chapter 7: C# .NET Programming](#)

[Chapter 8: Epic Prototypes, Classic Projects, Historic Genre](#)

[Chapter 9: Research and Case Studies](#)

The Album is available in French as well at Album.NeceMoon.com (or necemonyai.com/Blog/page/L-Album.aspx)

Conclusion

This is Good Bye. However, we can stay in touch. Feel very welcome to add me on [LinkedIn](#), [Twitter](#) or [Facebook](#). If you also have any tools and tactics that help you achieve things efficiently, I would like you to tell me about that. My email address is necemon@gmail.com. You are more than welcome to write to me and report any possible mistake in this document, or to suggest any improvement, or to tell me about your favourite parts. However, if you don't like The Album, don't bother writing to me.

Anyways, I wish you all the best for your ongoing and next projects. Thank you for reading. And many thanks to [Wikipedia](#), [MSDN](#), [IconFinder](#) and [FreeDigitalPhotos](#) for the clarifications and the graphic resources. Special thanks to the Evasium Team. Thanks to all those who contributed, thank you Ahou The African Chick, Antoine Mian, Cyriac Gbogou, Darren Mart, Edith Brou, Holty Sow, Israel Yoroba, Jean Luc Houedanou, Jean-Patrick Ehouman, Karen Kakou, Nanda Seye, Nnenna Nwakanma, Olivier Madiba, Vanessa Lecosson and Yehni Djidji.

Thank you Monty Oum, rest in peace.

[Share The Album with Your Mates](#)

Just click on any relevant icon below. It only takes a couple of seconds and it's free for everyone.

Alternatively, you can share this link on any social media website or email it to your contacts that would benefit from reading it: <http://TheAlbum.NeceMoon.com>

