Skeptic's Introduction to Objective-C:

Ten Things I *Love* About ObjC

and

Fifteen Things I *Hate* About ObjC

Jonathan 'Wolf' Rentzsch
http://rentzsch.com

Who am I? Why am I here?

- I’m an ObjC Skeptic.
- I actively resisted learning ObjC
  - Platform limitations
  - Syntax
  - lack of Garbage Collection
- Now I’m coding in ObjC
  - but only “where it makes sense”
  - That’s the Skeptic/Pragmatist: I use and recommend ObjC, but I recognize its problems

Why are you here?

- I'm going to walk you through what I *love* and *hate* about ObjC.
- ObjC gets a bunch of stuff right.
- However, I'm not an ObjC apologist.
  - When something sucks, I'm going to point it out.
  - But most of the time I'll also explain why it had to suck. (Hint: it's usually C/C++ compatibility.)

:-)

Love

Love 1: Lightweight

- ObjC is simply C plus a preprocessor and a small runtime
- Not much different from plain C, so the learning curve is short.

Love 2: Compatible

- Since ObjC is just a fancier C, it's very compatible with C (and now C++) code bases.
- ~~Hint: when integrating with C++, you probably should use my NSXException package for ObjC/C++ exception bridging.~~
- New exception model (circa gcc 3.3/Panther) largely cures this.

Love 3: (Fairly Good) Runtime Object Model

- I’ve been waiting *forever* for real desktop object model
- COM is lame, and SOM was apparently too heavy.
- ObjC is here, and is actually being used.
- Highly pragmatic runtime: Simple and fast.
- Good metadata for introspection.
- Does *most* everything you want.
  - (will readdress in *Hate* point)

Love 4: Message Sending

- ObjC has real, fast, runtime message sending.
- Coupled with language's design, allows many uses of multiple inheritance to go away.

Love 5: Categories

- Allows addition of methods to another class
  - Even a class you don’t have source code for.
  - i.e. Add a “rot13” method to NSString.
- Also allows overriding of existing methods.
  - Good for fixing bugs, and for tracing execution.
  - Not without suckage, will readdress in *Hate* section.

Love 6: Posing

- Cool runtime magic.
- Allows your class to take over and “pose” as another class.
- Deep hacking potential.

Love 7: Key Value Coding

- Introspection allows easy binding of strings to object methods and/or variables.
- Latest Real Life™ Example:
- ORU_R01_PIDPD1NTEPV1PV2ORCOBRNTEOBXNTECTI.ORU_R01_PIDPD1NTEPV1PV2.0.PID.0.4.0.1
- This fetches the patient's first name from the insane HL7 medical-industry standard format.

Love 8: #import

- Saves the traditional preprocessor dance:
  #ifndef __ARMOR__
  #define __ARMOR__
  // code goes here
  #endif
- Why wasn't this part of the C/C++ standard like ten years ago?

Love 9: -classNamed

- NSBundle makes it easy to load a class by name.
- C++ makes this *very* hard for no good reason.

Love 10: Preprocessor

- ObjC has C's preprocessor.
- __FILE__, __LINE__ and __PRETTY_FUNCTION__ all are there.
- Conditional compilation rocks.
- Makes compile-time assertion generation control possible.

:-(

Hate

Hate 1: Syntax

- Thing I hate most, both politically and technically.
- Consistent point of argument:
  - Newbies usually are unsure, and learning ObjC on their own time. The weird syntax is just an obstacle.
  - Complaining about it invokes NeXTie insecurity/elitism, which turns off newbie even more so.
- **Might** be worth the political costs...
- **but it's barely any better!!**

Hate 1: Syntax cont'd

- Really two issues: keyed parameters and brace syntax.
- Keyed parameters:
  - Definitely a "**win**".
  - But ObjC doesn't use them nearly effectively enough:
    - should allow reordering of parameters
    - should allow per-parameter defaults
      - could win big over C++ here
    - Maybe make them optional (Python)

Hate 1: Syntax cont'd

- Brace syntax:
  - Do you really believe:
    [[[MyClass alloc] init:[foo bar]] autorelease]
  - is easier to read or write than, say:
    MyClass.alloc.init(foo.bar).autorelase
  - *I screwed up typing the first one, but not the second.*
  - This isn't Lisp — the brace syntax buys you nothing in terms of language tricks.
  - **Why?** *Language Grammar Integration*

Hate 2: Lightweight

- ObjC does little for you.
- Heavy use of coding idioms since:
  - the same code has to be repeated often
  - little language support for anything other than message sending
- ObjC guys think all this explicit coding is *good*. They say "*no magic*".
- There's a long road from the explicitness of ObjC to Perl. Making the common case more convenient isn't going to kill you. *Some* syntactic sugar is *good*.

Hate 2: Lightweight cont’d

- **Why?** *The ObjC folks will tell you “no magic”*
- I disagree. It has more to do with:
  - ObjC being a simple superset of C.
  - A mediocre language being propped-up by a great framework.

Hate 3: Pointers

- ObjC gets raw pointers from its C heritage.
- Raw pointers are evil and must be stopped.
- At least, we need thin wrappers over raw pointers.
- Raw pointers preclude good garbage collection.
- I had a real hard time justifying learning a new language that lacks garbage collection. That's coming from a guy who *knows* manual memory management.
- **Why?** *C/C++ compatibility*

Hate 4: Alloc/Init Dance

- Modern ObjC separates object *allocation* & *initialization*
- This is **not** wrong by itself. Indeed, this would be wrong if you **couldn’t**.
- It **is** wrong that all code everywhere must separately call both +alloc and -init, in the right order.
- Here, “no magic” == “more code” == “more bugs”.
- Except for the dumb syntax, I think C++’s new/placement new gets this right.
- **Why?** *No good reason (yes, I know about +new)*

Hate 5: Designated Initializers

- Back on the "no magic" meme, an ObjC class has a *designated initializer:* an initializer all other initializers should call through.
- But ObjC provides **zero** lingual support for this very important indicator.
- You're left with optional, nonstandard comments and/or heuristics indicating such an initializer.
- **Why?** *No good reason*

Hate 6: Initialization Idiom

- Reads like a wacko wrote it:

```
if( self = [super init] ) {
  // initialization code here
}
return self;
```

- *[super init]* is fine
- Assigning to *self* is wacko.
- Placing the assignment in the *if()* reads like a common error. Better compilers (CodeWarrior) will actually warn/error about this.

Hate 6: Init Idiom cont'd

- You **can** do this instead (recommended):

```
self = [super init];
if( self ) {
   // initialization code here
}
return self;
```

- **Why?** *Unlike C++, which an instance has only one "this" pointer, ObjC instances have multiple, scoped "selves."*

Hate 7: No Stack Objects

- Can’t allocate ObjC objects on the stack (anymore).
- Even when you could, the benefit wasn’t there since there was no destructors or guarantee your -dealloc would be called.
- I like C++’s resource initialization is acquisition (RIIA) idiom. Makes writing exception-safe software easier. But it requires stack-based objects and destructors, which are “magic”. Yeah, like compiler-generated stack management is “magic”.
- **Why?** *No good reason. (ObjC++ wrappers can do this)*

Hate 8: Getter/Setters

- Surprisingly hard to get right.
- You need to decide if your instance variable should be handled like a reference (which can be shared) or a value (which cannot). Most of the time you want value semantics, but implementation efficiency often makes folks choose reference semantics.
- Your decision effects how you write your getters/setters. Had to figure this out for myself, as I never found any good explanation of it.
- **Why?** *Largely due to threading & reference counting.*

Hate 9: Preprocessor

- Yes, this was also on the *Love* list.
- Terribly useful, but inherently evil.
- Like raw pointers, it must die...
  - ...but replaced with something safer!
  - Java gets this **very** wrong. It desperately needs conditional compilation.
- **Why?** *C/C++ compatibility*

Hate 10: Messaging nil

- Sending a message to nil does *basically* **nothing**.
- **Why?** *This greatly reduces the need for checking for null all the time. Less code == less bugs, right?* ;-)
- **But** it also does a great job of hiding real bugs.
- Accidently disconnected outlets in shipping apps are legend.
- Wouldn't be so bad if it were easy to make messaging nil scream. But it's hard to do, and it screams all the time since Cocoa messages nil as a matter of course.

Hate 11: Class Unloading

- Can't unload a class once it's been loaded.
- Eliminates a bunch of possible cool tricks.
- **Why?** *No good reason.*

Hate 12: Overriding

- Neither Categories nor Posing can add instance variables to the target objects.
- There *are* inefficient work-arounds. But come on, let's get a **real** metaobject protocol runtime going.
- Categories are broken for overriding. Overriding a method more than once leads to undefined behavior.
- **Why?** *C/C++ compatibility mostly*

Hate 13: No Namespaces

- Not a major gripe, but when you need it, it's real handy.
- **Why?** *Stems from C, but perhaps could piggyback on C++ namespace support.*

Hate 14: id should be id*

- not clear:
  ```
  NSString *foo = @"foo";
  id bar = foo;
  ```
- better:
  ```
  NSString *foo = @"foo";
  id *bar = foo;
  ```
- Here, the pointer assignment is more explicit.
- Strange for a language that is otherwise explicit about everything else.
- **Why?** No idea. *Maybe "code cleanliness"?*

Hate 15: NeXTie Arrogance

- The entire "if you're not 100% enthusiastic about ObjC syntax then you're stupid" gets real old, real fast
- **Why?**
  - *Insecurity.*
  - *Elitism.*
  - *Tired of always having to defend their language:*
    - *From other language users*
    - *And their own newbie ObjC users (like me!)*

:-/

Indifferent

Indifferent 1: Frameworks

- Can’t really use a completely alternative framework with ObjC.
- I’d care if Cocoa wasn’t the best one publicly available.
- I reserve the right to change my mind when Adobe open sources their framework ;-)

Indifferent 2: Memory

- Cocoa uses reference counting
- I can handle *manual* memory management and I can obviously handle *automatic* memory management
- Cocoa's is kinda-manual, kinda-auto.
  - This was a **big** stumbling block for me
  - Definitely would have been high on *Hate* list
  - Likened to car with auto transmission but w/ clutch
  - Then I learned how pervasive NSAutoreleasePool is
- So docs suck, but retain/release is okay. !good && !bad

Indifferent 3: Type Safety

- *This isn't going to go over well with this crowd* ;-)
- ObjC doesn't offer as much type safety as, say, C++.
- I'm a big fan of catching errors as early as possible.
- Formal Protocols help enough that I really don't care. I've only made one type error in a few months of Cocoa programming. And that was because I was doing undocumented things :-)
- My attitude may change once I pick up a type inferred language like O'Caml.