From C++ to Objective-C

This document is designed to act as a bridge between C++ and Objective-C.

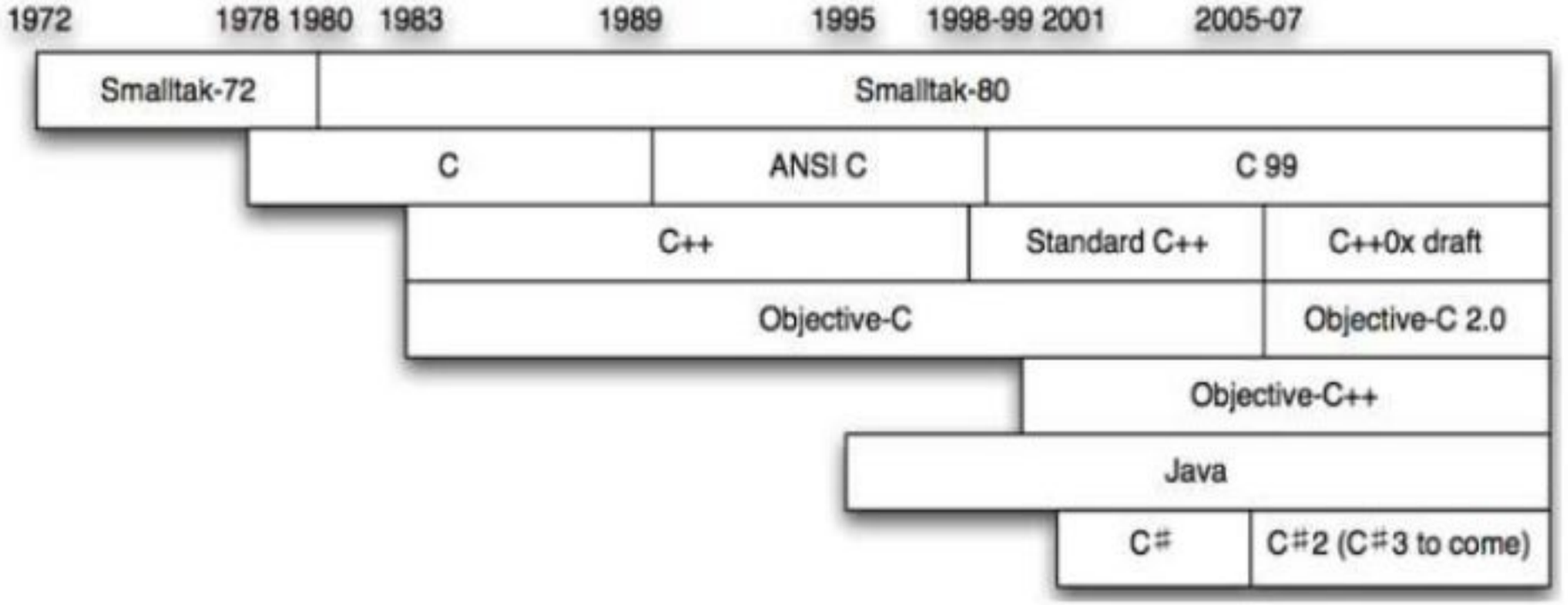
Why Objective-C?

solarwinds msp

```
#import "AppDelegate.h"
#import <ScriptingBridge/ScriptingBridge.h>
#import <objc/runtime.h>

NSString * const TerminalNotifierBundleID = @"fr.julienxx.oss.terminal-notifier";
NSString * const NotificationCenterUIBundleID = @"com.apple.notificationcenterui";

NSString * _fakeBundleIdentifier = nil;

@implementation NSBundle (FakeBundleIdentifier)

- (NSString *)__bundleIdentifier;
{
  if (self == [NSBundle mainBundle])
  {
    return _fakeBundleIdentifier ? _fakeBundleIdentifier : TerminalNotifierBundleID;
  }
  else
  {
    return [self __bundleIdentifier];
  }
}

@end
```

A short history of Objective-C

Timeline of Java, C, C#, C++ and Objective-C



Syntax overview

solarwinds msp

C++

```
//In file Foo.h

#pragma once //compilation guard
class Foo
{
   ...
};



//In file Foo.cpp
#include "Foo.h"
...
```

Objective-C

```
//In file Foo.h

//class declaration, different from
//the "interface" Java keyword
@interface Foo : NSObject
{
   ...
}
@end


//In file Foo.m
#import "Foo.h"
@implementation Foo

...

@end
```

Syntax overview

solarwinds msp

C++

```
class Foo
{
   double x;

   public:
      int   f(int x);
      float g(int x, int y);
};

int   Foo::f(int x) {...}
float Foo::g(int x, int y) {...}
```

Objective-C

```
@interface Foo : NSObject
{
   double x;
}

-(int)   f:(int)x;
-(float) g:(int)x :(int)y;

@end

@implementation Foo

-(int)   f:(int)x {...}
-(float) g:(int)x :(int)y {...}

@end
```

Syntax overview

solarwinds msp

C++

```
class Foo
{
   public:
      int x;
      int apple();

   protected:
      int y;
      int pear();

   private:
      int z;
      int banana();
};
```

Objective-C

```
@interface Foo : NSObject
{
   @public
      int x;

   @protected
      int y;

   @private
      int z;
}

-(int) apple;
-(int) pear;
-(int) banana;

@end
```

Root class, type id, nil and Nil values

Unlike C++, Objective-C defines a root class. Every new class should be a descendant of the root class. In Cocoa, that class is NSObject, and it provides a huge number of facilities for the run-time system.

```
OBJC_ROOT_CLASS
@interface NSObject <NSObject> {
  Class isa  OBJC_ISA_AVAILABILITY;
}

+ (void)load;

+ (void)initialize;
- (instancetype)init
+ (instancetype)new OBJC_SWIFT_UNAVAILABLE("use object initializers instead");
+ (instancetype)allocWithZone:(struct _NSZone *)zone OBJC_SWIFT_UNAVAILABLE("use object initializers instead");
+ (instancetype)alloc OBJC_SWIFT_UNAVAILABLE("use object initializers instead");
- (void)dealloc OBJC_SWIFT_UNAVAILABLE("use 'deinit' to define a de-initializer");

- (void)finalize OBJC_DEPRECATED("Objective-C garbage collection is no longer supported");

- (id)forwardingTargetForSelector:(SEL)aSelector OBJC_AVAILABLE(10.5, 2.0, 9.0, 1.0, 2.0);
- (void)forwardInvocation:(NSInvocation *)anInvocation OBJC_SWIFT_UNAVAILABLE("");
- (NSMethodSignature *)methodSignatureForSelector:(SEL)aSelector OBJC_SWIFT_UNAVAILABLE("");

+ (NSMethodSignature *)instanceMethodSignatureForSelector:(SEL)aSelector OBJC_SWIFT_UNAVAILABLE("")

+ (BOOL)resolveClassMethod:(SEL)sel OBJC_AVAILABLE(10.5, 2.0, 9.0, 1.0, 2.0);
+ (BOOL)resolveInstanceMethod:(SEL)sel OBJC_AVAILABLE(10.5, 2.0, 9.0, 1.0, 2.0);

+ (NSUInteger)hash;
+ (Class)superclass;
+ (Class)class OBJC_SWIFT_UNAVAILABLE("use 'aClass.self' instead");
+ (NSString *)description;
+ (NSString *)debugDescription;

@end
```

Root class, type id, nil and Nil values

"id" is a data type of object identifiers in Objective-C, which can be use for an object of any type no matter what class does it have. "id" is the final supertype of all objects.

By default, it is legal to send a message (call a method) to nil. The message is just ignored. The code can be greatly simplified by reducing the number of tests usually made with the null pointer.

```
NSString *name = @"dobegin";
id data = name;
NSURL *site = data;
NSString *host = site.host; // oops! runtime error!
// "unrecognized selector sent to instance"
```

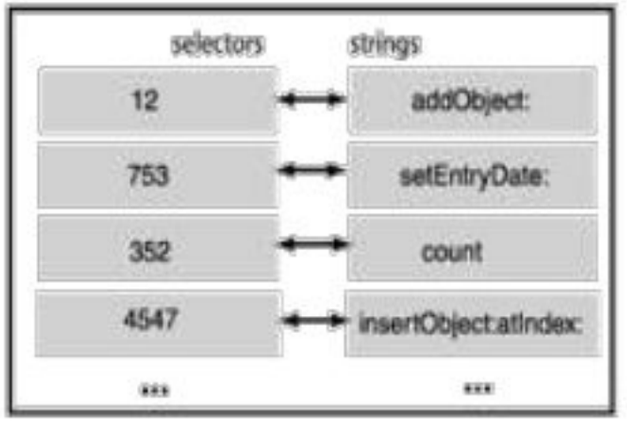
```
#if !defined(nil)
  #define nil (id)0
#endif

// For example, this expression...
if (name != nil && [name isEqualToString:@"Steve"]) { ... }

// ...can be simplified to:
if ([name isEqualToString:@"Steve"]) { ... }

#if !defined(Nil)
  #define Nil (Class)0
#endif

#inlcude <objc/runtime.h>

Class foo = objc_allocateClassPair([NSString class], "NSDictionary", 0);
if (foo != Nil)
{

}
```

Messages and transmission

A message is the name of a method, and any parameters associated with it, that are sent to, and executed by, an object. To get an object to do something, you send it a message telling it to apply a method.

- Search the class's method cache for the IMP
- If not found, search the class hierarchy for the IMP
- Jump to IMP if found (jmp assembly instruction)
- If not found, jump to _objc_msgforward
- All objects can forward messages they don't respond

```
id objc_msgSend(id receiver, SEL method, ...);
```




```
[receiver addObject: otherObject];

objc_msgSend(receiver, 12, otherObject);

addObject(receiver, otherObject);
```

Messages and transmission

In C++, code cannot be compiled if a method is called on an object that does not implement it. In Objective-C, there's a difference: one can always send a message to an object. If it can't be handled at run-time, it will be ignored (and raise an exception); moreover, instead of ignoring it, it can forward the message to another object.

```
-(void) forwardInvocation:(NSInvocation*)anInvocation
{
   //if we are here, that is because the object cannot handle
   //the message of the invocation
   //the bad selector can be obtained by sending "selector"
   //to the object "anInvocation"
   if ([anotherObject respondsToSelector:[anInvocation selector]])
      [anInvocation invokeWithTarget:anotherObject];
   else  //do not forget to try with the superclass
      [super forwardInvocation:anInvocation];
}
```

Messages and transmission

Delegation is common with user interface elements in Cocoa (tables, outlines. . .), taking advantage of the ability to send a message to an unknown object. An object can, for example, delegate some tasks to an assistant.

```
-(void) performHardWork:(id)task
{
   if ([assistant respondsToSelector:@selector(performHardWork:)])
      [assistant performHardWork:task];
   else
      [self findAnotherAssistant];
}
```

Protocols

A formal protocol is a set of methods that must be implemented in any conforming class. This can be seen as a certification regarding a class, ensuring that it is able to handle everything that is necessary for a given service. A class can conform to an unlimited number of protocols.

```
@protocol NSObject

- (BOOL)isEqual:(id)object;
@property (readonly) NSUInteger hash;

@property (readonly) Class superclass;
- (Class)class OBJC_SWIFT_UNAVAILABLE("use 'type(of: anObject)' instead");
- (instancetype)self;

- (id)performSelector:(SEL)aSelector;
- (id)performSelector:(SEL)aSelector withObject:(id)object;
- (id)performSelector:(SEL)aSelector withObject:(id)object1 withObject:(id)object2;

- (BOOL)isProxy;

- (BOOL)isKindOfClass:(Class)aClass;
- (BOOL)isMemberOfClass:(Class)aClass;
- (BOOL)conformsToProtocol:(Protocol *)aProtocol;

- (BOOL)respondsToSelector:(SEL)aSelector;

- (instancetype)retain OBJC_ARC_UNAVAILABLE;
- (oneway void)release OBJC_ARC_UNAVAILABLE;
- (instancetype)autorelease OBJC_ARC_UNAVAILABLE;
- (NSUInteger)retainCount OBJC_ARC_UNAVAILABLE;

- (struct _NSZone *)zone OBJC_ARC_UNAVAILABLE;

@property (readonly, copy) NSString *description;
@optional
@property (readonly, copy) NSString *debugDescription;

@end
```

Class categories

```
@interface NSArray : NSObject <NSCopying, NSMutableCopying, NSSecureCoding,
NSFastEnumeration>

@property (readonly) NSUInteger count;
- (ObjectType)objectAtIndex:(NSUInteger)index;
- (instancetype)init NS_DESIGNATED_INITIALIZER;
- (instancetype)initWithObjects:(const ObjectType _Nonnull [_Nullable])objects
count:(NSUInteger)cnt NS_DESIGNATED_INITIALIZER;
- (nullable instancetype)initWithCoder:(NSCoder *)aDecoder
NS_DESIGNATED_INITIALIZER;

@end
```

```
@interface NSArray(NSArrayCreation)

+ (instancetype)array;
+ (instancetype)arrayWithObject:(ObjectType)anObject;
+ (instancetype)arrayWithObjects:(const ObjectType _Nonnull [_Nonnull])objects
count:(NSUInteger)cnt;
+ (instancetype)arrayWithObjects:(ObjectType)firstObj, ...
NS_REQUIRES_NIL_TERMINATION;
+ (instancetype)arrayWithArray:(NSArray<ObjectType> *)array;

- (instancetype)initWithObjects:(ObjectType)firstObj, ...
NS_REQUIRES_NIL_TERMINATION;
- (instancetype)initWithArray:(NSArray<ObjectType> *)array;
- (instancetype)initWithArray:(NSArray<ObjectType> *)array copyItems:(BOOL)flag;

/* Reads array stored in NSPropertyList format from the specified url. */
- (nullable NSArray<ObjectType> *)initWithContentsOfURL:(NSURL *)url error:(NSError
**)error  API_AVAILABLE(macos(10.13), ios(11.0), watchos(4.0), tvos(11.0));
/* Reads array stored in NSPropertyList format from the specified url. */
+ (nullable NSArray<ObjectType> *)arrayWithContentsOfURL:(NSURL *)url
error:(NSError **)error API_AVAILABLE(macos(10.13), ios(11.0), watchos(4.0),
tvos(11.0)) NS_SWIFT_UNAVAILABLE("Use initializer instead");

@end
```

Properties

The notion of property can be met when defining classes. The keyword @property can be associated to a data member, to tell how the accessors can be automatically generated by the compiler. It aims at writing less code and save some development time.

A property is declared according to the following template :

@property *type name*;

or

@property(*attributes*) *type name*;

If they are not given, the attributes have a default value; otherwise, they can be redefined to answer the questions stated in the previous section. They can be :

- readwrite (default) or readonly to tell if the property should have both getter/setter or only the getter;
- assign (default), retain or copy, to tell how the value is stored internally;
- nonatomic to prevent thread-safety guards to be generated. They are generated by default (atomic).
- getter=..., setter=... to change the default name of the accessors.

```
@property (copy, nonatomic, nonnull) NSString *foo;
@property (copy, nonatomic, nonnull) NSString *str;
@property (strong, nonatomic, nullable) NSNumber *bar;
@property (copy, nonatomic, null_resettable) NSString *baz;
```

Working with Blocks

Block objects are a C-level syntactic and runtime feature. They are similar to standard C functions, but in addition to executable code they may also contain variable bindings to automatic (stack) or managed (heap) memory. A block can therefore maintain a set of state (data) that it can use to impact behavior when executed.

As a local variable:

```
returnType (^blockName)(parameterTypes) = ^returnType(parameters) {...};
```

As a property:

```
@property (nonatomic, copy, nullability) returnType (^blockName)(parameterTypes);
```

As a method parameter:

```
- (void)someMethodThatTakesABlock:(returnType (^nullability)(parameterTypes))blockName;
```

As an argument to a method call:

```
[someObject someMethodThatTakesABlock:^returnType (parameters) {...}];
```

As a typedef:

```
typedef returnType (^TypeName)(parameterTypes);
TypeName blockName = ^returnType(parameters) {...};
```

Working with Blocks

By default blocks are created on the stack. Meaning they only exist in the scope they have been created in.

```
//property
@property (nonatomic, copy) SomeBlockType someBlock;

//local variable
someBlockType someBlock = ^{
    NSLog(@"hi");
};
[someArray addObject:[someBlock copy]];
```

Blocks capture values from the enclosing scope.

```
__weak SomeObjectClass *weakSelf = self;

SomeBlockType someBlock = ^{
    SomeObjectClass *strongSelf = weakSelf;
    if (strongSelf) {
        [strongSelf someMethod];
    }
};
```

Working with Blocks

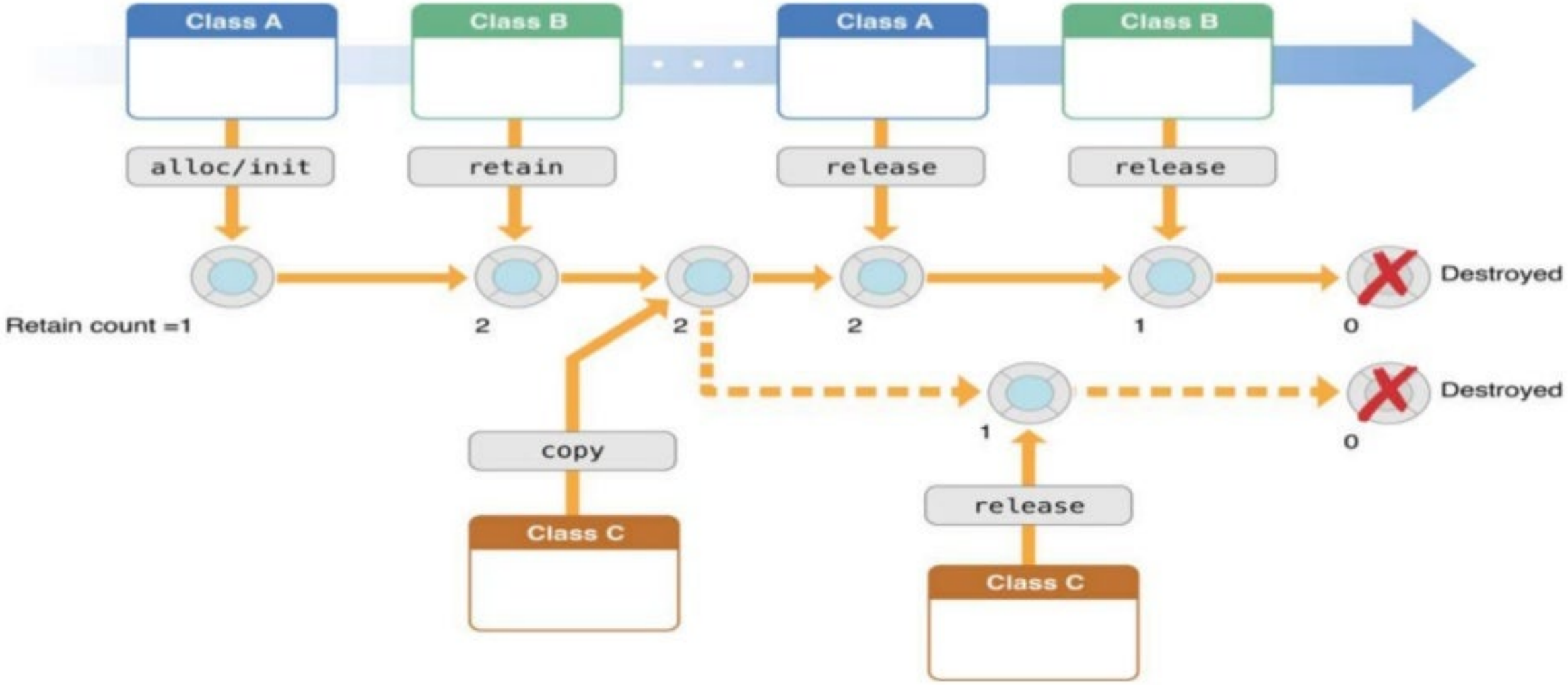
solarwinds msp

Use __block Variables to Share Storage

If you need to be able to change the value of a captured variable from within a block, you can use the __block storage type modifier on the original variable declaration. This means that the variable lives in storage that is shared between the lexical scope of the original variable and any blocks declared within that scope.

```
__block int anInteger = 42;

void (^testBlock)(void) = ^{
    NSLog(@"Integer is: %i", anInteger);
    anInteger = 100;
};

testBlock();
NSLog(@"Value of original variable is now: %i", anInteger);
```

Memory management



Basic Memory Management Rules

The memory management model is based on object ownership. Any object may have one or more owners. As long as an object has at least one owner, it continues to exist. If an object has no owners, the runtime system destroys it automatically. To make sure it is clear when you own an object and when you do not, Cocoa sets the following policy:

- **You own any object you create**
  You create an object using a method whose name begins with “alloc”, “new”, “copy”, or “mutableCopy”.
- **You can take ownership of an object using retain**
  A received object is normally guaranteed to remain valid within the method it was received in, and that method may also safely return the object to its invoker. You use retain in two situations: (1) In the implementation of an accessor method or an init method, to take ownership of an object you want to store as a property value; and (2) To prevent an object from being invalidated as a side-effect of some other operation.
- **When you no longer need it, you must relinquish ownership of an object you own**
  You relinquish ownership of an object by sending it a release message or an autorelease message. In Cocoa terminology, relinquishing ownership of an object is therefore typically referred to as “releasing” an object.
- **You must not relinquish ownership of an object you do not own**
  This is just corollary of the previous policy rules, stated explicitly.

Memory management

NSAutoreleasePool

An object that supports Cocoa's reference-counted memory management system. An autorelease pool stores objects that are sent a release message when the pool itself is drained. NSAutoreleasePool object contains objects that have received an autorelease message and when drained it sends a release message to each of those objects. Thus, sending autorelease instead of release to an object extends the lifetime of that object at least until the pool itself is drained (it may be longer if the object is subsequently retained). An object can be put into the same pool several times, in which case it receives a release message for each time it was put into the pool.

```
@autoreleasepool {
   // Code that creates autoreleased objects.
}
```

Like any other code block, autorelease pool blocks can be nested:

```
@autoreleasepool {
   // . . .
   @autoreleasepool {
      // . . .
   }
   . . .
}
```

Memory management

solarwinds msp

NSAutoreleasePool

Use local autorelease pool blocks to reduce peak memory footprint:

```
NSArray *urls = <# An array of file URLs #>;
for (NSURL *url in urls) {

  @autoreleasepool {
    NSError *error;
    NSString *fileContents = [NSString stringWithContentsOfURL:url
                                  encoding:NSUTF8StringEncoding error:&error];
    /* Process the string, creating and autoreleasing more objects. */
  }
}
```

Grand Central Dispatch

Grand Central Dispatch (GCD) is a technology developed by Apple Inc. to optimize application support for systems with multi-core processors and other symmetric multiprocessing systems. It is an implementation of task parallelism based on the thread pool pattern.

- *Dispatch Queues* are objects that maintain a queue of *tasks*, either anonymous code blocks or functions, and execute these tasks in their turn. The library automatically creates several queues with different priority levels that execute several tasks concurrently, selecting the optimal number of tasks to run based on the operating environment. A client to the library may also create any number of serial queues, which execute tasks in the order they are submitted, one at a time

```
- (IBAction)analyzeDocument:(NSButton *)sender {
    dispatch_async(dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0), ^{
        NSDictionary *stats = [myDoc analyze];
        dispatch_async(dispatch_get_main_queue(), ^{
            [myModel setDict:stats];
            [myStatsView setNeedsDisplay:YES];
        });
    });
}
```

Grand Central Dispatch

- *Dispatch Sources* are objects that allow the client to register blocks or functions to execute asynchronously upon system events, such as a socket or file descriptor being ready for reading or writing, or a POSIX signal.

```
dispatch_source_t CreateDispatchTimer(uint64_t interval,
                                      uint64_t leeway,
                                      dispatch_queue_t queue,
                                      dispatch_block_t block)
{
   dispatch_source_t timer = dispatch_source_create(DISPATCH_SOURCE_TYPE_TIMER,
                                                    0, 0, queue);
   if (timer)
   {
                                      dispatch_source_set_timer(timer, dispatch_walltime(NULL, 0), interval, leeway);
         dispatch_source_set_event_handler(timer, block);
         dispatch_resume(timer);
            }
   return timer;
}
```

Grand Central Dispatch

- *Dispatch Groups* are objects that allow several tasks to be grouped for later joining. Tasks can be added to a queue as a member of a group, and then the client can use the group object to wait until all of the tasks in that group have completed.

```
dispatch_queue_t queue = dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0);
dispatch_group_t group = dispatch_group_create();

// Add a task to the group
dispatch_group_async(group, queue, ^{
   // Some asynchronous work
});

// Do some other work while the tasks execute.

// When you cannot make any more forward progress,
// wait on the group to block the current thread.
dispatch_group_wait(group, DISPATCH_TIME_FOREVER);

// Release the group when it is no longer needed.
dispatch_release(group);
```

Grand Central Dispatch

- *Dispatch Semaphores* are objects that allow a client to permit only a certain number of tasks to execute concurrently.

```
// Create the semaphore, specifying the initial pool size
dispatch_semaphore_t fd_sema = dispatch_semaphore_create(getdtablesize() / 2);

// Wait for a free file descriptor
dispatch_semaphore_wait(fd_sema, DISPATCH_TIME_FOREVER);
fd = open("/etc/services", O_RDONLY);

// Release the file descriptor when done
close(fd);
dispatch_semaphore_signal(fd_sema);
```

Grand Central Dispatch

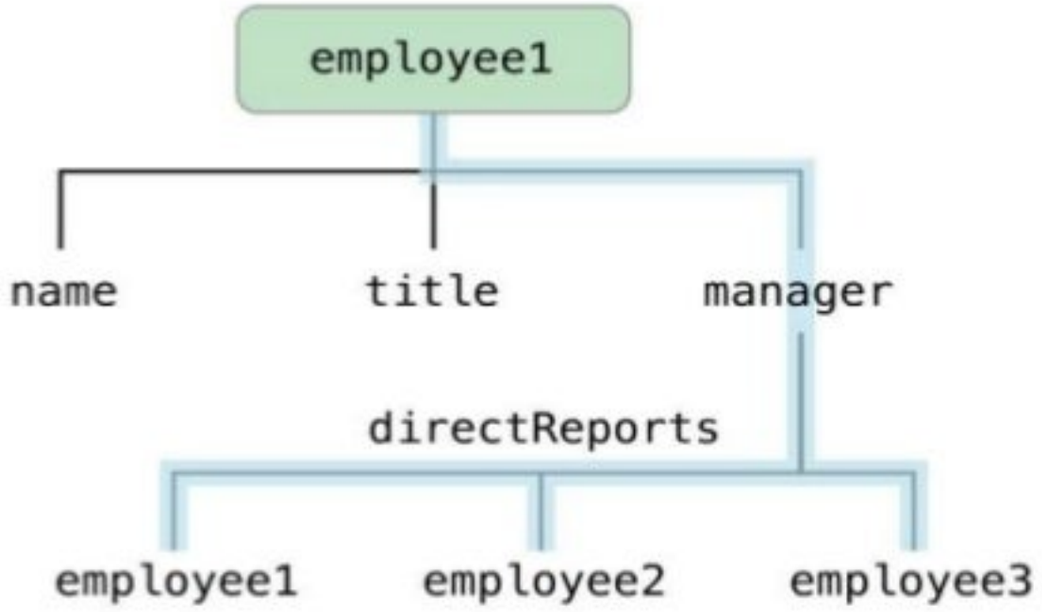
- Create singleton using GCD's dispatch_once

```
+ (instancetype)sharedInstance
{
   static dispatch_once_t once;
   static id sharedInstance;
   dispatch_once(&once, ^{
      sharedInstance = [[self alloc] init];
   });
   return sharedInstance;
}
```

Key-Value Coding

Key-value coding is a mechanism for indirectly accessing an object's attributes and relationships using string identifiers.

```
[employee1 setValue:newArray forKey:@"employee1.manager.directReports"];

NSArray *reports = [employee1 valueForKey:@"employee1.manager.directReports"];
```



employee1.manager.directReports

Key-Value Observing

Key-value observing is a mechanism that allows objects to be notified of changes to specified properties of other objects.

```
@interface MXParallaxView : UIView
@property (nonatomic,weak) MXParallaxHeader *parent;
@end

static void * const kMXParallaxHeaderKVOContext = (void*)&kMXParallaxHeaderKVOContext;

-(void)didMoveToSuperview
{
  [self.superview addObserver:self.parent forKeyPath:NSStringFromSelector(@selector(contentOffset)) options:NSKeyValueObservingOptionNew context:kMXParallaxHeaderKVOContext];
}

//This is where the magic happens...
-(void)observeValueForKeyPath:(NSString *)keyPath ofObject:(id)object change:(NSDictionary *)change context:(void *)context
{
  if (context == kMXParallaxHeaderKVOContext)
  {
    if ([keyPath isEqualToString:NSStringFromSelector(@selector(contentOffset))])
    {
    }
  }
  else
  {
    [super observeValueForKeyPath:keyPath ofObject:object change:change context:context];
  }
}
```

Objective-C Runtime

solarwinds msp

objc.h

```
typedef struct objc_class *Class;

typedef struct objc_object {
    Class isa;
} *id;

struct objc_class {
    Class isa;
};

typedef struct objc_selector *SEL;
typedef struct objc_method *Method;
typedef struct objc_ivar *Ivar;
typedef struct objc_category *Category;
typedef struct objc_property *objc_property_t;
```

Objective-C Runtime

solarwinds msp

Runtime library functions

- *Class manipulation*: class_addMethod, class_addIvar, class_replaceMethod.
- *Creating new classes*: class_allocateClassPair, class_registerClassPair.
- *Introspection*: class_getName, class_getSuperclass, class_getInstanceVariable, class_getProperty, class_copyMethodList, class_copyIvarList, class_copyPropertyList.

```
- (NSString *)description {
    NSMutableDictionary *propertyValues = [NSMutableDictionary dictionary];
    unsigned int propertyCount;
    objc_property_t *properties = class_copyPropertyList([self class], &propertyCount);
    for (unsigned int i = 0; i < propertyCount; i++) {
        char const *propertyName = property_getName(properties[i]);
        const char *attr = property_getAttributes(properties[i]);
        if (attr[1] == '@') {
            NSString *selector = [NSString stringWithCString:propertyName encoding:NSUTF8StringEncoding];
            SEL sel = sel_registerName([selector UTF8String]);
            NSObject * propertyValue = objc_msgSend(self, sel);
            propertyValues[selector] = propertyValue.description;
        }
    }
    free(properties);
    return [NSString stringWithFormat:@"%@:%@", self.class, propertyValues];
}
```

Objective-C Runtime

Runtime library functions

- *Object manipulation*: objc_msgSend, objc_getClass, object_copy.

```
@implementation NSMutableArray (CO)

+ (void)load {
    Method addObject = class_getInstanceMethod(self, @selector(addObject:));
    Method logAddObject = class_getInstanceMethod(self, @selector(logAddObject:));
    method_exchangeImplementations(addObject, logAddObject);
}

- (void)logAddObject:(id)aObject {
    [self logAddObject:aObject];
}

@end
```

Objective-C Runtime

Runtime library functions

- *Working with associative links*

```
@interface UITableView (Additions)

@property(nonatomic, strong) UIView *placeholderView;

@end

static char key;

@implementation UITableView (Additions)

-(void)setPlaceholderView:(UIView *)placeholderView {
   objc_setAssociatedObject(self, &key, placeholderView, OBJC_ASSOCIATION_RETAIN_NONATOMIC);
}

-(UIView *) placeholderView {
   return objc_getAssociatedObject(self, &key);
}

@end
```

Q&A

Thanks! Any questions, comments?

Alexander.Markevich@solarwinds.com