Objective-C 2.0

- A brief history
- What it looks like
- How it works
- The ties that bind

Objective-C 2.0

A brief history

Thank you Wikipedia

Objective-C .1

- Developed in the 1980's, by Brad Cox and Tom Love
- Add Smalltalk style Object Oriented Messaging to a compiled language
- Extensions to the C compiler
- Thin strict superset
- Tiny and fast runtime

Smalltalk ran on a virtual machine

Warts 'n' All!

Objective C 1.0

- Licensed and reimplemented by NeXT in 1988
- NeXT's advanced system libraries (OpenStep) made possible by dynamics of the language
- 1996 Apple acquired NeXT
- OpenStep became MacOSX

No Legacy Code

Steve Jobs took control

Objective C++

- Thin strict superset of C++

More Warts!

- Takes advantage of C++'s language features
- Some rules to obey for mixing OO features.

Do not cross the beams!

Objective C 2.0

- OpenStep remains advanced under continual development in MacOSX
- Language features added to complete with other languages and simplify development

Objective C 2.0

What it looks like

Verbose

CLASS MESSAGES

Classes are called Interfaces

Non-primitives inherit from NSObject

```
@interface Dishwasher : NSObject
{
// Private data definition
    NSString* model;
}

- (void) startAtTime: (NSTime*) time withDishes: (NSArray*) dishes andAddRinseAgent: (bool) addRinseAgent;

@end
```

Messages are a series of named parameters

Types are NOT part of the signature

Private methods need not be in public header file

```
@implementation Dishwasher

- (void) injectGallansOfWater: (float) gallons inSeconds: (float) seconds
{
}

- (void) startAtTime: (NSTime*) time withDishes: (NSArray*) dishes andAddRinseAgent: (bool) addRinseAgent
{
   [self injectGallansOfWater: 4.5 inSeconds: 20.0];
}

@end
```

Send a message

PROTOCOL ADHERENCE

```
@protocol Appliance

- (NSColor*) getColor;

@end
```

Interfaces are called Protocols

```
@interface Dishwasher : NSObject <Appliance>
{
// Private data definition
}

@end
```

Protocol Adherences are specified here

```
@implementation Dishwasher

- (NSColor*) getColor()
{
    return [NSColor colorWithComponents: 1.0, 1.0, 1.0];
}
@end
```

Optional implementation is a private implementation detail

Static Factory Methods are common

CATEGORIES

Reopen an interface

No Data Definition

```
@interface Appliance (DavesAppliance)

- (void) doEverything: (NSArray*) everything;

@end
```

Implement new method using no direct members

```
@implementation Dishwasher (DavesAppliance)

- (void) doEverything: (NSArray*) everything;
{
}

@end
```

Method injected at Runtime

```
int main()
{
    Appliance* myAppliance = [Dishwasher newDishwasher];
    [myAppliance doEverything];
}
```

Objective C 2.0

How it works

Rather Well

Messages

Don't go changing the types!

- Message Signatures are in a global pool
- Every Message has a compile time GUID
- Message sending uses GUID
- Mutable class message tables created at runtime
- Null objects or unknown (per instance) messages are dropped

Objects easily Swizzled

Forwarding Mechanism Available

ACCESSING LOW LEVEL RUNTIME

You can get a method's GUID

```
{
    SEL methodToExecute = @selector(doEverything);
    //SEL methodToExecute = sel_getUid("doEverything")
    [myAppliance performSelector: methodToExecute withObjects: everything];
}
```

And send the message yourself

There are many thread and scheduling options for performSelector

Every instance has methods to query the runtime

```
- (NSMethodSignature *)methodSignatureForSelector:(SEL)aSelector
- (IMP)methodForSelector:(SEL)aSelector
- (BOOL)respondsToSelector:(SEL)aSelector
```

Override this to forward unknown messages

```
- (retval_t) forward:(SEL) sel :(arglist_t) args
```

```
- (BOOL)isKindOfClass:(Class)aClass
- (BOOL)isMemberOfClass:(Class)aClass
```

Class is an opaque type

New Features in 2.0

64bit rewritten runtime

- Properties
  - Member access semantics with get/set as private implementation
  - Automatic bindings support
- Fast Enumeration
  - “For Each” on NSFastEnumeration protocol
- Protocol Implementations
  - Default implementations for protocol methods
- Protocol Enforcements
  - Required and Optional flags on methods

Poor Resource Management

PROPERTIES

```
@interface TrackDef : TrackElement {
@protected
   NSSize size;
   NSArray* anchors;
}

@property (nonatomic, readonly) NSString* name;
@property (nonatomic, readonly) NSSize size;
@property (nonatomic, readonly) NSArray* anchors;

@end
```

Synthesized properties must have data members

```
@implementation TrackDef

@synthesize size;
@synthesize anchors;

- (NSString*) name {
   return @"Obj-C Rocks!";
}

@end
```

Properties can have public getter and setter messages manufactured

```
NSString* theName = myTrackDef.name;
```

Objective-C 2.0

The ties that bind

Making better apps

Automatic Bindings

- All support for dynamic bindings included in every instance
- Compiler aware of binding's informal protocols
- Introspection methods built in for bindings
- Persistable
- Simple Informal Protocol
  - Grammar support for property transformations and aggregation

NSKEYVALUECODING

Getting Values

- valueForKey:
- valueForKeyPath:
- dictionaryWithValuesForKeys:
- valueForUndefinedKey:
- mutableArrayValueForKey:
- mutableArrayValueForKeyPath:
- mutableSetValueForKey:
- mutableSetValueForKeyPath:

Setting Values

- setValue:forKeyPath:
- setValuesForKeysWithDictionary:
- setNilValueForKey:
- setValue:forKey:
- setValue:forUndefinedKey:

Changing Default Behavior

+ accessInstanceVariablesDirectly

Validation

- validateValue:forKey:error:
- validateValue:forKeyPath:error:

Runtime Property Queries

NSObject adheres to this Protocol

- NSKeyValueCoding is NSObject's Protocol for Property introspection

Informal Protocol

A Set of Property Names as part of a Contract used for dynamic and loose bindings - often used for plugins

- iChat Video Effects
- Screen Savers
- Music Visualizers
- etc

No Code is Good Code

- Bindings specified by string paths
- String paths may contain transformation and aggregate functions
- Bindings persisted with objects
- UI is persistable object map

NO CODE GENERATION!!!

Demonstrate “Codeless” Application...