Objective-C Style Guide

Core Principles

- Efficiency: Optimize for performance on Apple platforms.
- Readability: Use clear, descriptive names that follow Cocoa conventions.
- Safety: Minimize the risk of crashes and memory leaks.

Naming Conventions

- Prefixes: Use 3-letter prefixes for class names to avoid namespace collisions (e.g., `GTMUser`).
- Classes: `UpperCamelCase`.
- Methods: `lowerCamelCase`. Use colon-separated descriptive segments. `- (void)updateUserWithName:(NSString *)name age:(NSInteger)age;`
- Variables and Properties: `lowerCamelCase`.
- Constants: `k` prefix followed by `UpperCamelCase`. `kGlobalTimeout`.

Memory Management (ARC)

- Always use Automatic Reference Counting (ARC).
- Ownership: Use `strong` for ownership and `weak` to avoid retain cycles (essential for delegates).
- Safety: Use `copy` for classes with mutable counterparts (like `NSString`, `NSArray`).
- Assign: Use `assign` for primitive types (NSInteger, CGFloat).

Properties and Instance Variables

- Prefer properties (`@property`) over direct instance variable access.
- Non-atomic: Use `nonatomic` by default unless true thread safety is required.
- Access: Always use `self.property` to ensure the correct getter/setter logic is executed.
- Dot Notation: Use dot notation for property access, but method notation for actions.

Nullability Annotations

Essential for Swift interoperability.

- `_Nullable`: The value can be nil.
- `_Nonnull`: The value must not be nil.
- `_Null_unspecified`: Default state (avoid if possible).

Modern Syntax (Literals)

- Arrays: `@[ @1, @2 ]` instead of `[NSArray arrayWith...]`.
- Dictionaries: `@{ @"key" : @"value" }`.
- Numbers: `@42`, `@YES`.

Error Handling

- Always use `NSError **` for functions that might fail.
- Check the return value (usually a BOOL) before inspecting the error object.
- Never crash purposefully; handle errors gracefully.

Blocks

- Use `typedef` for complex block signatures.
- Guard against retain cycles: Use a `__weak` reference to `self` inside blocks. `__weak typeof(self) weakSelf = self;`

Categories and Extensions

- Categories: Use to add methods to existing classes. Name methods with a prefix to avoid collisions.
- Class Extensions: Use in the `.m` file to declare private properties and methods.

Best Practices

Init and Dealloc

- Modern `init`: Always check if `self = [super init]` is non-nil before proceeding.
- `dealloc`: Use only for cleaning up non-ARC resources (like C pointers or removing observers).

Protocols

Use protocols for delegation and data-source patterns. This promotes loose coupling between components.

Documentation

Use HeaderDoc or Doxygen-style comments (`/** ... */`) to document public headers.