```
#import "AFURLSessionManager.h"
#import <objc/runtime.h>
#ifndef NSFoundationNumber
#define NSFoundationNumber
#else
#define NSFoundationNumber NSFoundationNumber
#endif
static dispatch_queue_t url_session_manager_creation_queue() {
    static dispatch_queue_t af_url_session_manager_creation_queue;
    static dispatch_once_t onceToken;
    dispatch_once(&onceToken, ^{
        af_url_session_manager_creation_queue = dispatch_queue_create("",
DISPATCH_QUEUE_SERIAL);
    });
    return af_url_session_manager_creation_queue;
}
static void url_session_manager_create_task_safely(dispatch_block_t block) {
    if (NSFoundationNumber <
NSFoundationNumber_With_Fixed_5871104061079552_bug) {
        dispatch_sync(url_session_manager_creation_queue(), block);
    } else {
        block();
    }
}
static dispatch_queue_t url_session_manager_processing_queue() {
    static dispatch_queue_t af_url_session_manager_processing_queue;
    static dispatch_once_t onceToken;
    dispatch_once(&onceToken, ^{
        af_url_session_manager_processing_queue = dispatch_queue_create("",
DISPATCH_QUEUE_CONCURRENT);
    });
    return af_url_session_manager_processing_queue;
}
static dispatch_group_t url_session_manager_completion_group() {
    static dispatch_group_t af_url_session_manager_completion_group;
    static dispatch_once_t onceToken;
    dispatch_once(&onceToken, ^{
        af_url_session_manager_completion_group = dispatch_group_create();
    });
    return af_url_session_manager_completion_group;
}
NSString * const AFNetworkingTaskDidResumeNotification = @"";
NSString * const AFNetworkingTaskDidCompleteNotification = @"";
NSString * const AFNetworkingTaskDidSuspendNotification = @"";
NSString * const AFURLSessionDidInvalidateNotification = @"";
NSString * const AFURLSessionDownloadTaskDidFailToMoveFileNotification = @"";
NSString * const AFNetworkingTaskDidCompleteSerializedResponseKey = @"";
NSString * const AFNetworkingTaskDidCompleteResponseSerializerKey = @"";
NSString * const AFNetworkingTaskDidCompleteResponseDataKey = @"";
NSString * const AFNetworkingTaskDidCompleteErrorKey = @"";
```

```
NSString * const AFNetworkingTaskDidCompleteAssetPathKey = @"";
static NSString * const AFURLSessionManagerLockName = @"";
static NSUInteger const
AFMaximumNumberOfAttemptsToRecreateBackgroundSessionUploadTask = 3;
typedef void (^AFURLSessionDidBecomeInvalidBlock)(NSURLSession *session, NSError
*error);
typedef NSURLSessionAuthChallengeDisposition
(^AFURLSessionDidReceiveAuthenticationChallengeBlock)(NSURLSession *session,
NSURLAuthenticationChallenge *challenge, NSURLCredential * __autoreleasing
*credential);
typedef NSURLRequest *
(^AFURLSessionTaskWillPerformRedirectionBlock)(NSURLSession *session,
NSURLSessionTask *task, NSURLResponse *response, NSURLRequest *request);
typedef NSURLSessionAuthChallengeDisposition
(^AFURLSessionTaskDidReceiveAuthenticationChallengeBlock)(NSURLSession
*session, NSURLSessionTask *task, NSURLAuthenticationChallenge *challenge,
NSURLCredential * __autoreleasing *credential);
typedef void
(^AFURLSessionDidFinishEventsForBackgroundURLSessionBlock)(NSURLSession
*session);
typedef NSInputStream * (^AFURLSessionTaskNeedNewBodyStreamBlock)(NSURLSession
*session, NSURLSessionTask *task);
typedef void (^AFURLSessionTaskDidSendBodyDataBlock)(NSURLSession *session,
NSURLSessionTask *task, int64_t bytesSent, int64_t totalBytesSent, int64_t
totalBytesExpectedToSend);
typedef void (^AFURLSessionTaskDidCompleteBlock)(NSURLSession *session,
NSURLSessionTask *task, NSError *error);
typedef NSURLSessionResponseDisposition
(^AFURLSessionDataTaskDidReceiveResponseBlock)(NSURLSession *session,
NSURLSessionDataTask *dataTask, NSURLResponse *response);
typedef void (^AFURLSessionDataTaskDidBecomeDownloadTaskBlock)(NSURLSession
*session, NSURLSessionDataTask *dataTask, NSURLSessionDownloadTask
*downloadTask);
typedef void (^AFURLSessionDataTaskDidReceiveDataBlock)(NSURLSession *session,
NSURLSessionDataTask *dataTask, NSData *data);
typedef NSCachedURLResponse *
(^AFURLSessionDataTaskWillCacheResponseBlock)(NSURLSession *session,
NSURLSessionDataTask *dataTask, NSCachedURLResponse *proposedResponse);
typedef NSURL *
(^AFURLSessionDownloadTaskDidFinishDownloadingBlock)(NSURLSession *session,
NSURLSessionDownloadTask *downloadTask, NSURL *location);
typedef void (^AFURLSessionDownloadTaskDidWriteDataBlock)(NSURLSession
*session, NSURLSessionDownloadTask *downloadTask, int64_t bytesWritten, int64_t
totalBytesWritten, int64_t totalBytesExpectedToWrite);
typedef void (^AFURLSessionDownloadTaskDidResumeBlock)(NSURLSession *session,
NSURLSessionDownloadTask *downloadTask, int64_t fileOffset, int64_t
expectedTotalBytes);
typedef void (^AFURLSessionTaskProgressBlock)(NSProgress *);
typedef void (^AFURLSessionTaskCompletionHandler)(NSURLResponse *response, id
responseObject, NSError *error);
```

```
#pragma mark -
@interface AFURLSessionManagerTaskDelegate : NSObject <NSURLSessionTaskDelegate,
NSURLSessionDataDelegate, NSURLSessionDownloadDelegate>
- (instancetype)initWithTask:(NSURLSessionTask *)task;
@property (nonatomic, weak) AFURLSessionManager *manager;
@property (nonatomic, strong) NSMutableData *mutableData;
@property (nonatomic, strong) NSProgress *uploadProgress;
@property (nonatomic, strong) NSProgress *downloadProgress;
@property (nonatomic, copy) NSURL *downloadFileURL;
@property (nonatomic, copy) AFURLSessionDownloadTaskDidFinishDownloadingBlock
downloadTaskDidFinishDownloading;
@property (nonatomic, copy) AFURLSessionTaskProgressBlock uploadProgressBlock;
@property (nonatomic, copy) AFURLSessionTaskProgressBlock downloadProgressBlock;
@property (nonatomic, copy) AFURLSessionTaskCompletionHandler completionHandler;
@end
@implementation AFURLSessionManagerTaskDelegate
- (instancetype)initWithTask:(NSURLSessionTask *)task {
    self = [super init];
    if (!self) {
        return nil;
    }
       _mutableData = [NSMutableData data];
    _uploadProgress = [[NSProgress alloc] initWithParent:nil userInfo:nil];
    _downloadProgress = [[NSProgress alloc] initWithParent:nil userInfo:nil];
       __weak __typeof__(task) weakTask = task;
    for (NSProgress *progress in @[ _uploadProgress, _downloadProgress ])
    {
        progress.totalUnitCount = NSURLSessionTransferSizeUnknown;
        progress.cancellable = YES;
        progress.cancellationHandler = ^{
            [weakTask cancel];
        };
        progress.pausable = YES;
        progress.pausingHandler = ^{
            [weakTask suspend];
        };
#if AF_CAN_USE_AT_AVAILABLE
        if (@available(iOS 9, macOS 10.11, *))
#else
        if ([progress respondsToSelector:@selector(setResumingHandler:)])
#endif
        {
            progress.resumingHandler = ^{
                [weakTask resume];
            };
        }
               [progress addObserver:self

forKeyPath:NSStringFromSelector(@selector(fractionCompleted))
                     options:NSKeyValueObservingOptionNew
```

```
                        context:NULL];
    }
    return self;
}
- (void)dealloc {
    [self.downloadProgress removeObserver:self
forKeyPath:NSStringFromSelector(@selector(fractionCompleted))];
    [self.uploadProgress removeObserver:self
forKeyPath:NSStringFromSelector(@selector(fractionCompleted))];
}
#pragma mark - NSProgress Tracking
- (void)observeValueForKeyPath:(NSString *)keyPath ofObject:(id)object
change:(NSDictionary<NSString *,id> *)change context:(void *)context {
   if ([object isEqual:self.downloadProgress]) {
        if (self.downloadProgressBlock) {
            self.downloadProgressBlock(object);
        }
    }
    else if ([object isEqual:self.uploadProgress]) {
        if (self.uploadProgressBlock) {
            self.uploadProgressBlock(object);
        }
    }
}
#pragma mark - NSURLSessionTaskDelegate
- (void)URLSession:(__unused NSURLSession *)session
              task:(NSURLSessionTask *)task
didCompleteWithError:(NSError *)error
{
    __strong AFURLSessionManager *manager = self.manager;
    __block id responseObject = nil;
    __block NSMutableDictionary *userInfo = [NSMutableDictionary dictionary];
    userInfo[AFNetworkingTaskDidCompleteResponseSerializerKey] =
manager.responseSerializer;
    //基于区块链的可靠加权投票系统V1.0
    NSData *data = nil;
    if (self.mutableData) {
        data = [self.mutableData copy];
        //我们不再需要引用，所以把它去掉以获得一些内存.
        self.mutableData = nil;
    }
    if (self.downloadFileURL) {
        userInfo[AFNetworkingTaskDidCompleteAssetPathKey] =
self.downloadFileURL;
    } else if (data) {
        userInfo[AFNetworkingTaskDidCompleteResponseDataKey] = data;
    }
    if (error) {
        userInfo[AFNetworkingTaskDidCompleteErrorKey] = error;
        dispatch_group_async(manager.completionGroup ?:
```

```
url_session_manager_completion_group(), manager.completionQueue ?:
dispatch_get_main_queue(), ^{
            if (self.completionHandler) {
                self.completionHandler(task.response, responseObject, error);
            }
            dispatch_async(dispatch_get_main_queue(), ^{
                [[NSNotificationCenter defaultCenter]
postNotificationName:AFNetworkingTaskDidCompleteNotification object:task
userInfo:userInfo];
            });
        });
    } else {
        dispatch_async(url_session_manager_processing_queue(), ^{
            NSError *serializationError = nil;
            responseObject = [manager.responseSerializer
responseObjectForResponse:task.response data:data error:&serializationError];
            if (self.downloadFileURL) {
                responseObject = self.downloadFileURL;
            }
            if (responseObject) {
                userInfo[AFNetworkingTaskDidCompleteSerializedResponseKey] =
responseObject;
            }
            if (serializationError) {
                userInfo[AFNetworkingTaskDidCompleteErrorKey] =
serializationError;
            }
            dispatch_group_async(manager.completionGroup ?:
url_session_manager_completion_group(), manager.completionQueue ?:
dispatch_get_main_queue(), ^{
                if (self.completionHandler) {
                    self.completionHandler(task.response, responseObject,
serializationError);
                }
                dispatch_async(dispatch_get_main_queue(), ^{
                    [[NSNotificationCenter defaultCenter]
postNotificationName:AFNetworkingTaskDidCompleteNotification object:task
userInfo:userInfo];
                });
            });
        });
    }
}
#pragma mark - NSURLSessionDataDelegate
- (void)URLSession:(__unused NSURLSession *)session
          dataTask:(__unused NSURLSessionDataTask *)dataTask
    didReceiveData:(NSData *)data
{
    self.downloadProgress.totalUnitCount =
dataTask.countOfBytesExpectedToReceive;
```

```
    self.downloadProgress.completedUnitCount = dataTask.countOfBytesReceived;
    [self.mutableData appendData:data];
}
- (void)URLSession:(NSURLSession *)session task:(NSURLSessionTask *)task
   didSendBodyData:(int64_t)bytesSent
    totalBytesSent:(int64_t)totalBytesSent
totalBytesExpectedToSend:(int64_t)totalBytesExpectedToSend{
       self.uploadProgress.totalUnitCount = task.countOfBytesExpectedToSend;
    self.uploadProgress.completedUnitCount = task.countOfBytesSent;
}
#pragma mark - NSURLSessionDownloadDelegate
- (void)URLSession:(NSURLSession *)session
downloadTask:(NSURLSessionDownloadTask *)downloadTask
      didWriteData:(int64_t)bytesWritten
 totalBytesWritten:(int64_t)totalBytesWritten
totalBytesExpectedToWrite:(int64_t)totalBytesExpectedToWrite{
       self.downloadProgress.totalUnitCount = totalBytesExpectedToWrite;
    self.downloadProgress.completedUnitCount = totalBytesWritten;
}
- (void)URLSession:(NSURLSession *)session
downloadTask:(NSURLSessionDownloadTask *)downloadTask
 didResumeAtOffset:(int64_t)fileOffset
expectedTotalBytes:(int64_t)expectedTotalBytes{
       self.downloadProgress.totalUnitCount = expectedTotalBytes;
    self.downloadProgress.completedUnitCount = fileOffset;
}
- (void)URLSession:(NSURLSession *)session
      downloadTask:(NSURLSessionDownloadTask *)downloadTask
didFinishDownloadingToURL:(NSURL *)location
{
    self.downloadFileURL = nil;
    if (self.downloadTaskDidFinishDownloading) {
        self.downloadFileURL = self.downloadTaskDidFinishDownloading(session,
downloadTask, location);
        if (self.downloadFileURL) {
            NSError *fileManagerError = nil;
            if (![[NSFileManager defaultManager] moveItemAtURL:location
toURL:self.downloadFileURL error:&fileManagerError]) {
                [[NSNotificationCenter defaultCenter]
postNotificationName:AFURLSessionDownloadTaskDidFailToMoveFileNotification
object:downloadTask userInfo:fileManagerError.userInfo];
            }
        }
    }
}
@end
#pragma mark -
/**
 *  观察NSURLSessionTask的“state”的键值相关问题的解决方法`.
 *
```

```
 *  See:
 */
static inline void af_swizzleSelector(Class theClass, SEL originalSelector, SEL
swizzledSelector) {
    Method originalMethod = class_getInstanceMethod(theClass, originalSelector);
    Method swizzledMethod = class_getInstanceMethod(theClass, swizzledSelector);
    method_exchangeImplementations(originalMethod, swizzledMethod);
}
static inline BOOL af_addMethod(Class theClass, SEL selector, Method method) {
    return class_addMethod(theClass, selector,
method_getImplementation(method),  method_getTypeEncoding(method));
}
static NSString * const AFNSURLSessionTaskDidResumeNotification  = @"";
static NSString * const AFNSURLSessionTaskDidSuspendNotification = @"";
@interface _AFURLSessionTaskSwizzling : NSObject
@end
@implementation _AFURLSessionTaskSwizzling
+ (void)load {
    /**
     基于区块链的可靠加权投票系统V1.0
     s://github.com/AFNetworking/AFNetworking/pull/2702
     */
    if (NSClassFromString(@"NSURLSessionTask")) {
        NSURLSessionConfiguration *configuration = [NSURLSessionConfiguration
ephemeralSessionConfiguration];
        NSURLSession * session = [NSURLSession
sessionWithConfiguration:configuration];
#pragma GCC diagnostic push
#pragma GCC diagnostic ignored "-Wnonnull"
        NSURLSessionDataTask *localDataTask = [session dataTaskWithURL:nil];
#pragma clang diagnostic pop
        IMP originalAFResumeIMP =
method_getImplementation(class_getInstanceMethod([self class],
@selector(af_resume)));
        Class currentClass = [localDataTask class];
               while (class_getInstanceMethod(currentClass, @selector(resume)))
{
            Class superClass = [currentClass superclass];
            IMP classResumeIMP =
method_getImplementation(class_getInstanceMethod(currentClass,
@selector(resume)));
            IMP superclassResumeIMP =
method_getImplementation(class_getInstanceMethod(superClass,
@selector(resume)));
            if (classResumeIMP != superclassResumeIMP &&
                originalAFResumeIMP != classResumeIMP) {
                [self swizzleResumeAndSuspendMethodForClass:currentClass];
            }
            currentClass = [currentClass superclass];
        }
```

```
                [localDataTask cancel];
        [session finishTasksAndInvalidate];
    }
}
+ (void)swizzleResumeAndSuspendMethodForClass:(Class)theClass {
    Method afResumeMethod = class_getInstanceMethod(self, @selector(af_resume));
    Method afSuspendMethod = class_getInstanceMethod(self,
@selector(af_suspend));
    if (af_addMethod(theClass, @selector(af_resume), afResumeMethod)) {
        af_swizzleSelector(theClass, @selector(resume), @selector(af_resume));
    }
    if (af_addMethod(theClass, @selector(af_suspend), afSuspendMethod)) {
        af_swizzleSelector(theClass, @selector(suspend),
@selector(af_suspend));
    }
}
- (NSURLSessionTaskState)state {
    NSAssert(NO, @"State method should never be called in the actual dummy class");
    return NSURLSessionTaskStateCanceling;
}
- (void)af_resume {
    NSAssert([self respondsToSelector:@selector(state)], @"Does not respond to
state");
    NSURLSessionTaskState state = [self state];
    [self af_resume];
       if (state != NSURLSessionTaskStateRunning) {
        [[NSNotificationCenter defaultCenter]
postNotificationName:AFNSURLSessionTaskDidResumeNotification object:self];
    }
}
- (void)af_suspend {
    NSAssert([self respondsToSelector:@selector(state)], @"Does not respond to
state");
    NSURLSessionTaskState state = [self state];
    [self af_suspend];
       if (state != NSURLSessionTaskStateSuspended) {
        [[NSNotificationCenter defaultCenter]
postNotificationName:AFNSURLSessionTaskDidSuspendNotification object:self];
    }
}
@end
#pragma mark -
@interface AFURLSessionManager ()
@property (readwrite, nonatomic, strong) NSURLSessionConfiguration
*sessionConfiguration;
@property (readwrite, nonatomic, strong) NSOperationQueue *operationQueue;
@property (readwrite, nonatomic, strong) NSURLSession *session;
@property (readwrite, nonatomic, strong) NSMutableDictionary
*mutableTaskDelegatesKeyedByTaskIdentifier;
@property (readonly, nonatomic, copy) NSString *taskDescriptionForSessionTasks;
```

```
@property (readwrite, nonatomic, strong) NSLock *lock;
@property (readwrite, nonatomic, copy) AFURLSessionDidBecomeInvalidBlock
sessionDidBecomeInvalid;
@property (readwrite, nonatomic, copy)
AFURLSessionDidReceiveAuthenticationChallengeBlock
sessionDidReceiveAuthenticationChallenge;
@property (readwrite, nonatomic, copy)
AFURLSessionDidFinishEventsForBackgroundURLSessionBlock
didFinishEventsForBackgroundURLSession AF_API_UNAVAILABLE(macos);
@property (readwrite, nonatomic, copy)
AFURLSessionTaskWillPerformRedirectionBlock taskWillPerformRedirection;
@property (readwrite, nonatomic, copy)
AFURLSessionTaskDidReceiveAuthenticationChallengeBlock
taskDidReceiveAuthenticationChallenge;
@property (readwrite, nonatomic, copy) AFURLSessionTaskNeedNewBodyStreamBlock
taskNeedNewBodyStream;
@property (readwrite, nonatomic, copy) AFURLSessionTaskDidSendBodyDataBlock
taskDidSendBodyData;
@property (readwrite, nonatomic, copy) AFURLSessionTaskDidCompleteBlock
taskDidComplete;
@property (readwrite, nonatomic, copy)
AFURLSessionDataTaskDidReceiveResponseBlock dataTaskDidReceiveResponse;
@property (readwrite, nonatomic, copy)
AFURLSessionDataTaskDidBecomeDownloadTaskBlock dataTaskDidBecomeDownloadTask;
@property (readwrite, nonatomic, copy) AFURLSessionDataTaskDidReceiveDataBlock
dataTaskDidReceiveData;
@property (readwrite, nonatomic, copy)
AFURLSessionDataTaskWillCacheResponseBlock dataTaskWillCacheResponse;
@property (readwrite, nonatomic, copy)
AFURLSessionDownloadTaskDidFinishDownloadingBlock
downloadTaskDidFinishDownloading;
@property (readwrite, nonatomic, copy) AFURLSessionDownloadTaskDidWriteDataBlock
downloadTaskDidWriteData;
@property (readwrite, nonatomic, copy) AFURLSessionDownloadTaskDidResumeBlock
downloadTaskDidResume;
@end
@implementation AFURLSessionManager
- (instancetype)init {
    return [self initWithSessionConfiguration:nil];
}
- (instancetype)initWithSessionConfiguration:(NSURLSessionConfiguration
*)configuration {
    self = [super init];
    if (!self) {
        return nil;
    }
    if (!configuration) {
        configuration = [NSURLSessionConfiguration
defaultSessionConfiguration];
    }
```

```
    self.sessionConfiguration = configuration;
    self.operationQueue = [[NSOperationQueue alloc] init];
    self.operationQueue.maxConcurrentOperationCount = 1;
    self.session = [NSURLSession
sessionWithConfiguration:self.sessionConfiguration delegate:self
delegateQueue:self.operationQueue];
    self.responseSerializer = [AFJSONResponseSerializer serializer];
    self.securityPolicy = [AFSecurityPolicy defaultPolicy];
#if !TARGET_OS_WATCH
    self.reachabilityManager = [AFNetworkReachabilityManager sharedManager];
#endif
    self.mutableTaskDelegatesKeyedByTaskIdentifier = [[NSMutableDictionary
alloc] init];
    self.lock = [[NSLock alloc] init];
    self.lock.name = AFURLSessionManagerLockName;
    [self.session getTasksWithCompletionHandler:^(NSArray *dataTasks, NSArray
*uploadTasks, NSArray *downloadTasks) {
        for (NSURLSessionDataTask *task in dataTasks) {
            [self addDelegateForDataTask:task uploadProgress:nil
downloadProgress:nil completionHandler:nil];
        }
        for (NSURLSessionUploadTask *uploadTask in uploadTasks) {
            [self addDelegateForUploadTask:uploadTask progress:nil
completionHandler:nil];
        }
        for (NSURLSessionDownloadTask *downloadTask in downloadTasks) {
            [self addDelegateForDownloadTask:downloadTask progress:nil
destination:nil completionHandler:nil];
        }
    }];
    return self;
}
- (void)dealloc {
    [[NSNotificationCenter defaultCenter] removeObserver:self];
}
#pragma mark -
- (NSString *)taskDescriptionForSessionTasks {
    return [NSString stringWithFormat:@"%p", self];
}
- (void)taskDidResume:(NSNotification *)notification {
    NSURLSessionTask *task = notification.object;
    if ([task respondsToSelector:@selector(taskDescription)]) {
        if ([task.taskDescription
isEqualToString:self.taskDescriptionForSessionTasks]) {
            dispatch_async(dispatch_get_main_queue(), ^{
                [[NSNotificationCenter defaultCenter]
postNotificationName:AFNetworkingTaskDidResumeNotification object:task];
            });
        }
    }
```

```
}
- (void)taskDidSuspend:(NSNotification *)notification {
    NSURLSessionTask *task = notification.object;
    if ([task respondsToSelector:@selector(taskDescription)]) {
        if ([task.taskDescription
isEqualToString:self.taskDescriptionForSessionTasks]) {
            dispatch_async(dispatch_get_main_queue(), ^{
                [[NSNotificationCenter defaultCenter]
postNotificationName:AFNetworkingTaskDidSuspendNotification object:task];
            });
        }
    }
}
#pragma mark -
- (AFURLSessionManagerTaskDelegate *)delegateForTask:(NSURLSessionTask *)task {
    NSParameterAssert(task);
    AFURLSessionManagerTaskDelegate *delegate = nil;
    [self.lock lock];
    delegate =
self.mutableTaskDelegatesKeyedByTaskIdentifier[@(task.taskIdentifier)];
    [self.lock unlock];
    return delegate;
}
- (void)setDelegate:(AFURLSessionManagerTaskDelegate *)delegate
            forTask:(NSURLSessionTask *)task
{
    NSParameterAssert(task);
    NSParameterAssert(delegate);
    [self.lock lock];
    self.mutableTaskDelegatesKeyedByTaskIdentifier[@(task.taskIdentifier)] =
delegate;
    [self addNotificationObserverForTask:task];
    [self.lock unlock];
}
- (void)addDelegateForDataTask:(NSURLSessionDataTask *)dataTask
                uploadProgress:(nullable void (^)(NSProgress *uploadProgress))
uploadProgressBlock
              downloadProgress:(nullable void (^)(NSProgress
*downloadProgress)) downloadProgressBlock
             completionHandler:(void (^)(NSURLResponse *response, id
responseObject, NSError *error))completionHandler
{
    AFURLSessionManagerTaskDelegate *delegate =
[[AFURLSessionManagerTaskDelegate alloc] initWithTask:dataTask];
    delegate.manager = self;
    delegate.completionHandler = completionHandler;
    dataTask.taskDescription = self.taskDescriptionForSessionTasks;
    [self setDelegate:delegate forTask:dataTask];
    delegate.uploadProgressBlock = uploadProgressBlock;
    delegate.downloadProgressBlock = downloadProgressBlock;
```

```
}
- (void)addDelegateForUploadTask:(NSURLSessionUploadTask *)uploadTask
                        progress:(void (^)(NSProgress *uploadProgress))
uploadProgressBlock
               completionHandler:(void (^)(NSURLResponse *response, id
responseObject, NSError *error))completionHandler
{
    AFURLSessionManagerTaskDelegate *delegate =
[[AFURLSessionManagerTaskDelegate alloc] initWithTask:uploadTask];
    delegate.manager = self;
    delegate.completionHandler = completionHandler;
    uploadTask.taskDescription = self.taskDescriptionForSessionTasks;
    [self setDelegate:delegate forTask:uploadTask];
    delegate.uploadProgressBlock = uploadProgressBlock;
}
- (void)addDelegateForDownloadTask:(NSURLSessionDownloadTask *)downloadTask
                          progress:(void (^)(NSProgress *downloadProgress))
downloadProgressBlock
                       destination:(NSURL * (^)(NSURL *targetPath, NSURLResponse
*response))destination
                 completionHandler:(void (^)(NSURLResponse *response, NSURL
*filePath, NSError *error))completionHandler
{
    AFURLSessionManagerTaskDelegate *delegate =
[[AFURLSessionManagerTaskDelegate alloc] initWithTask:downloadTask];
    delegate.manager = self;
    delegate.completionHandler = completionHandler;
    if (destination) {
        delegate.downloadTaskDidFinishDownloading = ^NSURL * (NSURLSession *
__unused session, NSURLSessionDownloadTask *task, NSURL *location) {
            return destination(location, task.response);
        };
    }
    downloadTask.taskDescription = self.taskDescriptionForSessionTasks;
    [self setDelegate:delegate forTask:downloadTask];
    delegate.downloadProgressBlock = downloadProgressBlock;
}
- (void)removeDelegateForTask:(NSURLSessionTask *)task {
    NSParameterAssert(task);
    [self.lock lock];
    [self removeNotificationObserverForTask:task];
    [self.mutableTaskDelegatesKeyedByTaskIdentifier
removeObjectForKey:@(task.taskIdentifier)];
    [self.lock unlock];
}
#pragma mark -
- (NSArray *)tasksForKeyPath:(NSString *)keyPath {
    __block NSArray *tasks = nil;
    dispatch_semaphore_t semaphore = dispatch_semaphore_create(0);
    [self.session getTasksWithCompletionHandler:^(NSArray *dataTasks, NSArray
```

```
*uploadTasks, NSArray *downloadTasks) {
        if ([keyPath
isEqualToString:NSStringFromSelector(@selector(dataTasks))]) {
            tasks = dataTasks;
        } else if ([keyPath
isEqualToString:NSStringFromSelector(@selector(uploadTasks))]) {
            tasks = uploadTasks;
        } else if ([keyPath
isEqualToString:NSStringFromSelector(@selector(downloadTasks))]) {
            tasks = downloadTasks;
        } else if ([keyPath
isEqualToString:NSStringFromSelector(@selector(tasks))]) {
            tasks = [@[dataTasks, uploadTasks, downloadTasks]
valueForKeyPath:@"@unionOfArrays.self"];
        }
        dispatch_semaphore_signal(semaphore);
    }];
    dispatch_semaphore_wait(semaphore, DISPATCH_TIME_FOREVER);
    return tasks;
}
- (NSArray *)tasks {
    return [self tasksForKeyPath:NSStringFromSelector(_cmd)];
}
- (NSArray *)dataTasks {
    return [self tasksForKeyPath:NSStringFromSelector(_cmd)];
}
- (NSArray *)uploadTasks {
    return [self tasksForKeyPath:NSStringFromSelector(_cmd)];
}
- (NSArray *)downloadTasks {
    return [self tasksForKeyPath:NSStringFromSelector(_cmd)];
}
#pragma mark -
- (void)invalidateSessionCancelingTasks:(BOOL)cancelPendingTasks {
    if (cancelPendingTasks) {
        [self.session invalidateAndCancel];
    } else {
        [self.session finishTasksAndInvalidate];
    }
}
#pragma mark -
- (void)setResponseSerializer:(id
<AFURLResponseSerialization>)responseSerializer {
    NSParameterAssert(responseSerializer);
    _responseSerializer = responseSerializer;
}
#pragma mark -
- (void)addNotificationObserverForTask:(NSURLSessionTask *)task {
    [[NSNotificationCenter defaultCenter] addObserver:self
selector:@selector(taskDidResume:)
```

```
name:AFNSURLSessionTaskDidResumeNotification object:task];
    [[NSNotificationCenter defaultCenter] addObserver:self
selector:@selector(taskDidSuspend:)
name:AFNSURLSessionTaskDidSuspendNotification object:task];
}
- (void)removeNotificationObserverForTask:(NSURLSessionTask *)task {
    [[NSNotificationCenter defaultCenter] removeObserver:self
name:AFNSURLSessionTaskDidSuspendNotification object:task];
    [[NSNotificationCenter defaultCenter] removeObserver:self
name:AFNSURLSessionTaskDidResumeNotification object:task];
}
#pragma mark -
- (NSURLSessionDataTask *)dataTaskWithRequest:(NSURLRequest *)request
                            completionHandler:(void (^)(NSURLResponse
*response, id responseObject, NSError *error))completionHandler
{
    return [self dataTaskWithRequest:request uploadProgress:nil
downloadProgress:nil completionHandler:completionHandler];
}
- (NSURLSessionDataTask *)dataTaskWithRequest:(NSURLRequest *)request
                              uploadProgress:(nullable void (^)(NSProgress
*uploadProgress)) uploadProgressBlock
                            downloadProgress:(nullable void (^)(NSProgress
*downloadProgress)) downloadProgressBlock
                            completionHandler:(nullable void (^)(NSURLResponse
*response, id _Nullable responseObject,  NSError * _Nullable
error))completionHandler {
    __block NSURLSessionDataTask *dataTask = nil;
    url_session_manager_create_task_safely(^{
        dataTask = [self.session dataTaskWithRequest:request];
    });
    [self addDelegateForDataTask:dataTask uploadProgress:uploadProgressBlock
downloadProgress:downloadProgressBlock completionHandler:completionHandler];
    return dataTask;
}
#pragma mark -
- (NSURLSessionUploadTask *)uploadTaskWithRequest:(NSURLRequest *)request
                                         fromFile:(NSURL *)fileURL
                                         progress:(void (^)(NSProgress
*uploadProgress)) uploadProgressBlock
                                completionHandler:(void (^)(NSURLResponse
*response, id responseObject, NSError *error))completionHandler
{
    __block NSURLSessionUploadTask *uploadTask = nil;
    url_session_manager_create_task_safely(^{
        uploadTask = [self.session uploadTaskWithRequest:request
fromFile:fileURL];
            // uploadTask may be nil on iOS7 because
uploadTaskWithRequest:fromFile: may return nil despite being documented as nonnull
(s://devforums.apple.com/message/926113#926113)
```

```
        if (!uploadTask &&
self.attemptsToRecreateUploadTasksForBackgroundSessions &&
self.session.configuration.identifier) {
            for (NSUInteger attempts = 0; !uploadTask && attempts <
AFMaximumNumberOfAttemptsToRecreateBackgroundSessionUploadTask; attempts++) {
                uploadTask = [self.session uploadTaskWithRequest:request
fromFile:fileURL];
            }
        }
    });
       if (uploadTask) {
        [self addDelegateForUploadTask:uploadTask
                              progress:uploadProgressBlock
                     completionHandler:completionHandler];
    }
    return uploadTask;
}
- (NSURLSessionUploadTask *)uploadTaskWithRequest:(NSURLRequest *)request
                                         fromData:(NSData *)bodyData
                                         progress:(void (^)(NSProgress
*uploadProgress)) uploadProgressBlock
                                completionHandler:(void (^)(NSURLResponse
*response, id responseObject, NSError *error))completionHandler
{
    __block NSURLSessionUploadTask *uploadTask = nil;
    url_session_manager_create_task_safely(^{
        uploadTask = [self.session uploadTaskWithRequest:request
fromData:bodyData];
    });
    [self addDelegateForUploadTask:uploadTask progress:uploadProgressBlock
completionHandler:completionHandler];
    return uploadTask;
}
- (NSURLSessionUploadTask *)uploadTaskWithStreamedRequest:(NSURLRequest
*)request
                                                 progress:(void (^)(NSProgress
*uploadProgress)) uploadProgressBlock
                                        completionHandler:(void
(^)(NSURLResponse *response, id responseObject, NSError
*error))completionHandler
{
    __block NSURLSessionUploadTask *uploadTask = nil;
    url_session_manager_create_task_safely(^{
        uploadTask = [self.session uploadTaskWithStreamedRequest:request];
    });
    [self addDelegateForUploadTask:uploadTask progress:uploadProgressBlock
completionHandler:completionHandler];
    return uploadTask;
}
#pragma mark -
```

```
- (NSURLSessionDownloadTask *)downloadTaskWithRequest:(NSURLRequest *)request
                                             progress:(void (^)(NSProgress
*downloadProgress)) downloadProgressBlock
                                          destination:(NSURL * (^)(NSURL
*targetPath, NSURLResponse *response))destination
                                    completionHandler:(void (^)(NSURLResponse
*response, NSURL *filePath, NSError *error))completionHandler
{
    __block NSURLSessionDownloadTask *downloadTask = nil;
    url_session_manager_create_task_safely(^{
        downloadTask = [self.session downloadTaskWithRequest:request];
    });
    [self addDelegateForDownloadTask:downloadTask
progress:downloadProgressBlock destination:destination
completionHandler:completionHandler];
    return downloadTask;
}
- (NSURLSessionDownloadTask *)downloadTaskWithResumeData:(NSData *)resumeData
                                                progress:(void (^)(NSProgress
*downloadProgress)) downloadProgressBlock
                                             destination:(NSURL * (^)(NSURL
*targetPath, NSURLResponse *response))destination
                                       completionHandler:(void
(^)(NSURLResponse *response, NSURL *filePath, NSError *error))completionHandler
{
    __block NSURLSessionDownloadTask *downloadTask = nil;
    url_session_manager_create_task_safely(^{
        downloadTask = [self.session downloadTaskWithResumeData:resumeData];
    });
    [self addDelegateForDownloadTask:downloadTask
progress:downloadProgressBlock destination:destination
completionHandler:completionHandler];
    return downloadTask;
}
#pragma mark -
- (NSProgress *)uploadProgressForTask:(NSURLSessionTask *)task {
    return [[self delegateForTask:task] uploadProgress];
}
- (NSProgress *)downloadProgressForTask:(NSURLSessionTask *)task {
    return [[self delegateForTask:task] downloadProgress];
}
#pragma mark -
- (void)setSessionDidBecomeInvalidBlock:(void (^)(NSURLSession *session, NSError
*error))block {
    self.sessionDidBecomeInvalid = block;
}
-
(void)setSessionDidReceiveAuthenticationChallengeBlock:(NSURLSessionAuthChalle
ngeDisposition (^)(NSURLSession *session, NSURLAuthenticationChallenge
*challenge, NSURLCredential * __autoreleasing *credential))block {
```

```
    self.sessionDidReceiveAuthenticationChallenge = block;
}
#if !TARGET_OS_OSX
- (void)setDidFinishEventsForBackgroundURLSessionBlock:(void (^)(NSURLSession
*session))block {
    self.didFinishEventsForBackgroundURLSession = block;
}
#endif
#pragma mark -
- (void)setTaskNeedNewBodyStreamBlock:(NSInputStream * (^)(NSURLSession
*session, NSURLSessionTask *task))block {
    self.taskNeedNewBodyStream = block;
}
- (void)setTaskWillPerformRedirectionBlock:(NSURLRequest * (^)(NSURLSession
*session, NSURLSessionTask *task, NSURLResponse *response, NSURLRequest
*request))block {
    self.taskWillPerformRedirection = block;
}
-
(void)setTaskDidReceiveAuthenticationChallengeBlock:(NSURLSessionAuthChallenge
Disposition (^)(NSURLSession *session, NSURLSessionTask *task,
NSURLAuthenticationChallenge *challenge, NSURLCredential * __autoreleasing
*credential))block {
    self.taskDidReceiveAuthenticationChallenge = block;
}
- (void)setTaskDidSendBodyDataBlock:(void (^)(NSURLSession *session,
NSURLSessionTask *task, int64_t bytesSent, int64_t totalBytesSent, int64_t
totalBytesExpectedToSend))block {
    self.taskDidSendBodyData = block;
}
- (void)setTaskDidCompleteBlock:(void (^)(NSURLSession *session,
NSURLSessionTask *task, NSError *error))block {
    self.taskDidComplete = block;
}
#pragma mark -
- (void)setDataTaskDidReceiveResponseBlock:(NSURLSessionResponseDisposition
(^)(NSURLSession *session, NSURLSessionDataTask *dataTask, NSURLResponse
*response))block {
    self.dataTaskDidReceiveResponse = block;
}
- (void)setDataTaskDidBecomeDownloadTaskBlock:(void (^)(NSURLSession *session,
NSURLSessionDataTask *dataTask, NSURLSessionDownloadTask *downloadTask))block {
    self.dataTaskDidBecomeDownloadTask = block;
}
- (void)setDataTaskDidReceiveDataBlock:(void (^)(NSURLSession *session,
NSURLSessionDataTask *dataTask, NSData *data))block {
    self.dataTaskDidReceiveData = block;
}
- (void)setDataTaskWillCacheResponseBlock:(NSCachedURLResponse *
(^)(NSURLSession *session, NSURLSessionDataTask *dataTask, NSCachedURLResponse
```

```
*proposedResponse))block {
    self.dataTaskWillCacheResponse = block;
}
#pragma mark -
- (void)setDownloadTaskDidFinishDownloadingBlock:(NSURL * (^)(NSURLSession
*session, NSURLSessionDownloadTask *downloadTask, NSURL *location))block {
    self.downloadTaskDidFinishDownloading = block;
}
- (void)setDownloadTaskDidWriteDataBlock:(void (^)(NSURLSession *session,
NSURLSessionDownloadTask *downloadTask, int64_t bytesWritten, int64_t
totalBytesWritten, int64_t totalBytesExpectedToWrite))block {
    self.downloadTaskDidWriteData = block;
}
- (void)setDownloadTaskDidResumeBlock:(void (^)(NSURLSession *session,
NSURLSessionDownloadTask *downloadTask, int64_t fileOffset, int64_t
expectedTotalBytes))block {
    self.downloadTaskDidResume = block;
}
#pragma mark - NSObject
- (NSString *)description {
    return [NSString stringWithFormat:@"<%@: %p, session: %@, operationQueue:
%@>", NSStringFromClass([self class]), self, self.session, self.operationQueue];
}
- (BOOL)respondsToSelector:(SEL)selector {
    if (selector ==
@selector(URLSession:task:willPerformRedirection:newRequest:completionHandler:
)) {
        return self.taskWillPerformRedirection != nil;
    } else if (selector ==
@selector(URLSession:dataTask:didReceiveResponse:completionHandler:)) {
        return self.dataTaskDidReceiveResponse != nil;
    } else if (selector ==
@selector(URLSession:dataTask:willCacheResponse:completionHandler:)) {
        return self.dataTaskWillCacheResponse != nil;
    }
#if !TARGET_OS_OSX
    else if (selector ==
@selector(URLSessionDidFinishEventsForBackgroundURLSession:)) {
        return self.didFinishEventsForBackgroundURLSession != nil;
    }
#endif
    return [[self class] instancesRespondToSelector:selector];
}
#pragma mark - NSURLSessionDelegate
- (void)URLSession:(NSURLSession *)session
didBecomeInvalidWithError:(NSError *)error
{
    if (self.sessionDidBecomeInvalid) {
        self.sessionDidBecomeInvalid(session, error);
    }
```

```
    [[NSNotificationCenter defaultCenter]
postNotificationName:AFURLSessionDidInvalidateNotification object:session];
}
- (void)URLSession:(NSURLSession *)session
didReceiveChallenge:(NSURLAuthenticationChallenge *)challenge
 completionHandler:(void (^)(NSURLSessionAuthChallengeDisposition disposition,
NSURLCredential *credential))completionHandler
{
    NSURLSessionAuthChallengeDisposition disposition =
NSURLSessionAuthChallengePerformDefaultHandling;
    __block NSURLCredential *credential = nil;
    if (self.sessionDidReceiveAuthenticationChallenge) {
        disposition = self.sessionDidReceiveAuthenticationChallenge(session,
challenge, &credential);
    } else {
        if ([challenge.protectionSpace.authenticationMethod
isEqualToString:NSURLAuthenticationMethodServerTrust]) {
            if ([self.securityPolicy
evaluateServerTrust:challenge.protectionSpace.serverTrust
forDomain:challenge.protectionSpace.host]) {
                credential = [NSURLCredential
credentialForTrust:challenge.protectionSpace.serverTrust];
                if (credential) {
                    disposition = NSURLSessionAuthChallengeUseCredential;
                } else {
                    disposition =
NSURLSessionAuthChallengePerformDefaultHandling;
                }
            } else {
                disposition =
NSURLSessionAuthChallengeCancelAuthenticationChallenge;
            }
        } else {
            disposition = NSURLSessionAuthChallengePerformDefaultHandling;
        }
    }
    if (completionHandler) {
        completionHandler(disposition, credential);
    }
}
#pragma mark - NSURLSessionTaskDelegate
- (void)URLSession:(NSURLSession *)session
              task:(NSURLSessionTask *)task
willPerformRedirection:(NSURLResponse *)response
        newRequest:(NSURLRequest *)request
 completionHandler:(void (^)(NSURLRequest *))completionHandler
{
    NSURLRequest *redirectRequest = request;
    if (self.taskWillPerformRedirection) {
        redirectRequest = self.taskWillPerformRedirection(session, task,
```

```
response, request);
    }
    if (completionHandler) {
        completionHandler(redirectRequest);
    }
}
- (void)URLSession:(NSURLSession *)session
              task:(NSURLSessionTask *)task
didReceiveChallenge:(NSURLAuthenticationChallenge *)challenge
 completionHandler:(void (^)(NSURLSessionAuthChallengeDisposition disposition,
NSURLCredential *credential))completionHandler
{
    NSURLSessionAuthChallengeDisposition disposition =
NSURLSessionAuthChallengePerformDefaultHandling;
    __block NSURLCredential *credential = nil;
    if (self.taskDidReceiveAuthenticationChallenge) {
        disposition = self.taskDidReceiveAuthenticationChallenge(session, task,
challenge, &credential);
    } else {
        if ([challenge.protectionSpace.authenticationMethod
isEqualToString:NSURLAuthenticationMethodServerTrust]) {
            if ([self.securityPolicy
evaluateServerTrust:challenge.protectionSpace.serverTrust
forDomain:challenge.protectionSpace.host]) {
                disposition = NSURLSessionAuthChallengeUseCredential;
                credential = [NSURLCredential
credentialForTrust:challenge.protectionSpace.serverTrust];
            } else {
                disposition =
NSURLSessionAuthChallengeCancelAuthenticationChallenge;
            }
        } else {
            disposition = NSURLSessionAuthChallengePerformDefaultHandling;
        }
    }
    if (completionHandler) {
        completionHandler(disposition, credential);
    }
}
- (void)URLSession:(NSURLSession *)session
              task:(NSURLSessionTask *)task
 needNewBodyStream:(void (^)(NSInputStream *bodyStream))completionHandler
{
    NSInputStream *inputStream = nil;
    if (self.taskNeedNewBodyStream) {
        inputStream = self.taskNeedNewBodyStream(session, task);
    } else if (task.originalRequest.BodyStream &&
[task.originalRequest.BodyStream conformsToProtocol:@protocol(NSCopying)]) {
        inputStream = [task.originalRequest.BodyStream copy];
    }
```

```
    if (completionHandler) {
        completionHandler(inputStream);
    }
}
- (void)URLSession:(NSURLSession *)session
              task:(NSURLSessionTask *)task
   didSendBodyData:(int64_t)bytesSent
    totalBytesSent:(int64_t)totalBytesSent
totalBytesExpectedToSend:(int64_t)totalBytesExpectedToSend
{
    int64_t totalUnitCount = totalBytesExpectedToSend;
    if(totalUnitCount == NSURLSessionTransferSizeUnknown) {
        NSString *contentLength = [task.originalRequest
valueForHeaderField:@"Content-Length"];
        if(contentLength) {
            totalUnitCount = (int64_t) [contentLength longLongValue];
        }
    }
       AFURLSessionManagerTaskDelegate *delegate = [self delegateForTask:task];
       if (delegate) {
        [delegate URLSession:session task:task didSendBodyData:bytesSent
totalBytesSent:totalBytesSent
totalBytesExpectedToSend:totalBytesExpectedToSend];
    }
    if (self.taskDidSendBodyData) {
        self.taskDidSendBodyData(session, task, bytesSent, totalBytesSent,
totalUnitCount);
    }
}
- (void)URLSession:(NSURLSession *)session
              task:(NSURLSessionTask *)task
didCompleteWithError:(NSError *)error
{
    AFURLSessionManagerTaskDelegate *delegate = [self delegateForTask:task];
    // delegate may be nil when completing a task in the background
    if (delegate) {
        [delegate URLSession:session task:task didCompleteWithError:error];
        [self removeDelegateForTask:task];
    }
    if (self.taskDidComplete) {
        self.taskDidComplete(session, task, error);
    }
}
#pragma mark - NSURLSessionDataDelegate
- (void)URLSession:(NSURLSession *)session
          dataTask:(NSURLSessionDataTask *)dataTask
didReceiveResponse:(NSURLResponse *)response
 completionHandler:(void (^)(NSURLSessionResponseDisposition
disposition))completionHandler
{
```

```
    NSURLSessionResponseDisposition disposition = NSURLSessionResponseAllow;
    if (self.dataTaskDidReceiveResponse) {
        disposition = self.dataTaskDidReceiveResponse(session, dataTask,
response);
    }
    if (completionHandler) {
        completionHandler(disposition);
    }
}
- (void)URLSession:(NSURLSession *)session
          dataTask:(NSURLSessionDataTask *)dataTask
didBecomeDownloadTask:(NSURLSessionDownloadTask *)downloadTask
{
    AFURLSessionManagerTaskDelegate *delegate = [self delegateForTask:dataTask];
    if (delegate) {
        [self removeDelegateForTask:dataTask];
        [self setDelegate:delegate forTask:downloadTask];
    }
    if (self.dataTaskDidBecomeDownloadTask) {
        self.dataTaskDidBecomeDownloadTask(session, dataTask, downloadTask);
    }
}
- (void)URLSession:(NSURLSession *)session
          dataTask:(NSURLSessionDataTask *)dataTask
    didReceiveData:(NSData *)data
{
    AFURLSessionManagerTaskDelegate *delegate = [self delegateForTask:dataTask];
    [delegate URLSession:session dataTask:dataTask didReceiveData:data];
    if (self.dataTaskDidReceiveData) {
        self.dataTaskDidReceiveData(session, dataTask, data);
    }
}
- (void)URLSession:(NSURLSession *)session
          dataTask:(NSURLSessionDataTask *)dataTask
 willCacheResponse:(NSCachedURLResponse *)proposedResponse
 completionHandler:(void (^)(NSCachedURLResponse
*cachedResponse))completionHandler
{
    NSCachedURLResponse *cachedResponse = proposedResponse;
    if (self.dataTaskWillCacheResponse) {
        cachedResponse = self.dataTaskWillCacheResponse(session, dataTask,
proposedResponse);
    }
    if (completionHandler) {
        completionHandler(cachedResponse);
    }
}
#if !TARGET_OS_OSX
- (void)URLSessionDidFinishEventsForBackgroundURLSession:(NSURLSession
*)session {
```

```
    if (self.didFinishEventsForBackgroundURLSession) {
        dispatch_async(dispatch_get_main_queue(), ^{
            self.didFinishEventsForBackgroundURLSession(session);
        });
    }
}
#endif
#pragma mark - NSURLSessionDownloadDelegate
- (void)URLSession:(NSURLSession *)session
      downloadTask:(NSURLSessionDownloadTask *)downloadTask
didFinishDownloadingToURL:(NSURL *)location
{
    AFURLSessionManagerTaskDelegate *delegate = [self
delegateForTask:downloadTask];
    if (self.downloadTaskDidFinishDownloading) {
        NSURL *fileURL = self.downloadTaskDidFinishDownloading(session,
downloadTask, location);
        if (fileURL) {
            delegate.downloadFileURL = fileURL;
            NSError *error = nil;
                        if (![[NSFileManager defaultManager]
moveItemAtURL:location toURL:fileURL error:&error]) {
                [[NSNotificationCenter defaultCenter]
postNotificationName:AFURLSessionDownloadTaskDidFailToMoveFileNotification
object:downloadTask userInfo:error.userInfo];
            }
            return;
        }
    }
    if (delegate) {
        [delegate URLSession:session downloadTask:downloadTask
didFinishDownloadingToURL:location];
    }
}
- (void)URLSession:(NSURLSession *)session
      downloadTask:(NSURLSessionDownloadTask *)downloadTask
      didWriteData:(int64_t)bytesWritten
 totalBytesWritten:(int64_t)totalBytesWritten
totalBytesExpectedToWrite:(int64_t)totalBytesExpectedToWrite
{
       AFURLSessionManagerTaskDelegate *delegate = [self
delegateForTask:downloadTask];
       if (delegate) {
        [delegate URLSession:session downloadTask:downloadTask
didWriteData:bytesWritten totalBytesWritten:totalBytesWritten
totalBytesExpectedToWrite:totalBytesExpectedToWrite];
    }
    if (self.downloadTaskDidWriteData) {
        self.downloadTaskDidWriteData(session, downloadTask, bytesWritten,
totalBytesWritten, totalBytesExpectedToWrite);
```

```
    }
}
- (void)URLSession:(NSURLSession *)session
      downloadTask:(NSURLSessionDownloadTask *)downloadTask
 didResumeAtOffset:(int64_t)fileOffset
expectedTotalBytes:(int64_t)expectedTotalBytes
{
       AFURLSessionManagerTaskDelegate *delegate = [self
delegateForTask:downloadTask];
       if (delegate) {
        [delegate URLSession:session downloadTask:downloadTask
didResumeAtOffset:fileOffset expectedTotalBytes:expectedTotalBytes];
    }
    if (self.downloadTaskDidResume) {
        self.downloadTaskDidResume(session, downloadTask, fileOffset,
expectedTotalBytes);
    }
}
#pragma mark - NSSecureCoding
+ (BOOL)supportsSecureCoding {
    return YES;
}
- (instancetype)initWithCoder:(NSCoder *)decoder {
    NSURLSessionConfiguration *configuration = [decoder
decodeObjectOfClass:[NSURLSessionConfiguration class]
forKey:@"sessionConfiguration"];
    self = [self initWithSessionConfiguration:configuration];
    if (!self) {
        return nil;
    }
    return self;
}
- (void)encodeWithCoder:(NSCoder *)coder {
    [coder encodeObject:self.session.configuration
forKey:@"sessionConfiguration"];
}
#pragma mark - NSCopying
- (instancetype)copyWithZone:(NSZone *)zone {
    return [[[self class] allocWithZone:zone]
initWithSessionConfiguration:self.session.configuration];
}
@end
#import "TApplicationException.h"
#import "TProtocolUtil.h"
#import "TObjective-C.h"
@implementation TApplicationException
- (id) initWithType: (int) type
             reason: (NSString *) reason
{
  mType = type;
```

```
  NSString * name;
  switch (type) {
  case TApplicationException_UNKNOWN_METHOD:
    name = @"Unknown method";
    break;
  case TApplicationException_INVALID_MESSAGE_TYPE:
    name = @"Invalid message type";
    break;
  case TApplicationException_WRONG_METHOD_NAME:
    name = @"Wrong method name";
    break;
  case TApplicationException_BAD_SEQUENCE_ID:
    name = @"Bad sequence ID";
    break;
  case TApplicationException_MISSING_RESULT:
    name = @"Missing result";
    break;
  case TApplicationException_INTERNAL_ERROR:
    name = @"Internal error";
    break;
  case TApplicationException_PROTOCOL_ERROR:
    name = @"Protocol error";
    break;
  case TApplicationException_INVALID_TRANSFORM:
    name = @"Invalid transform";
    break;
  case TApplicationException_INVALID_PROTOCOL:
    name = @"Invalid protocol";
    break;
  case TApplicationException_UNSUPPORTED_CLIENT_TYPE:
    name = @"Unsupported client type";
    break;
  default:
    name = @"Unknown";
    break;
  }
  self = [super initWithName: name reason: reason userInfo: nil];
  return self;
}
+ (TApplicationException *) read: (id <TProtocol>) protocol
{
  NSString * reason = nil;
  int type = TApplicationException_UNKNOWN;
  int fieldType;
  int fieldID;
  [protocol readStructBeginReturningName: NULL];
  while (true) {
    [protocol readFieldBeginReturningName: NULL
              type: &fieldType
              fieldID: &fieldID];
```

```
    if (fieldType == TType_STOP) {
      break;
    }
    switch (fieldID) {
    case 1:
      if (fieldType == TType_STRING) {
        reason = [protocol readString];
      } else {
        [TProtocolUtil skipType: fieldType onProtocol: protocol];
      }
      break;
    case 2:
      if (fieldType == TType_I32) {
        type = [protocol readI32];
      } else {
        [TProtocolUtil skipType: fieldType onProtocol: protocol];
      }
      break;
    default:
      [TProtocolUtil skipType: fieldType onProtocol: protocol];
      break;
    }
    [protocol readFieldEnd];
  }
  [protocol readStructEnd];
  return [TApplicationException exceptionWithType: type reason: reason];
}
- (void) write: (id <TProtocol>) protocol
{
  [protocol writeStructBeginWithName: @"TApplicationException"];
  if ([self reason] != nil) {
    [protocol writeFieldBeginWithName: @"message"
                 type: TType_STRING
                 fieldID: 1];
    [protocol writeString: [self reason]];
    [protocol writeFieldEnd];
  }
  [protocol writeFieldBeginWithName: @"type"
               type: TType_I32
               fieldID: 2];
  [protocol writeI32: mType];
  [protocol writeFieldEnd];
  [protocol writeFieldStop];
  [protocol writeStructEnd];
}
+ (TApplicationException *) exceptionWithType: (int) type
                                       reason: (NSString *) reason
{
  return [[[TApplicationException alloc] initWithType: type
                                          reason: reason] autorelease_stub];
```

```
}
@end
#import "TApplicationException.h"
#import "TProtocolUtil.h"
#import "TObjective-C.h"
@implementation TApplicationException
- (id) initWithType: (int) type
             reason: (NSString *) reason
{
  mType = type;
  NSString * name;
  switch (type) {
  case TApplicationException_UNKNOWN_METHOD:
    name = @"Unknown method";
    break;
  case TApplicationException_INVALID_MESSAGE_TYPE:
    name = @"Invalid message type";
    break;
  case TApplicationException_WRONG_METHOD_NAME:
    name = @"Wrong method name";
    break;
  case TApplicationException_BAD_SEQUENCE_ID:
    name = @"Bad sequence ID";
    break;
  case TApplicationException_BAD_SEQUENCE_ID:
    name = @"Bad sequence ID";
    break;
  case TApplicationException_MISSING_RESULT:
    name = @"Missing result";
    break;
  case TApplicationException_INTERNAL_ERROR:
    name = @"Internal error";
    break;
  case TApplicationException_PROTOCOL_ERROR:
    name = @"Protocol error";
    break;
  case TApplicationException_INVALID_TRANSFORM:
    name = @"Invalid transform";
    break;
  case TApplicationException_INVALID_PROTOCOL:
    name = @"Invalid protocol";
    break;
  case TApplicationException_UNSUPPORTED_CLIENT_TYPE:
    name = @"Unsupported client type";
    break;
  default:
    name = @"Unknown";
    break;
  }
  self = [super initWithName: name reason: reason userInfo: nil];
```

```
  return self;
}
+ (TApplicationException *) read: (id <TProtocol>) protocol
{
  NSString * reason = nil;
  int type = TApplicationException_UNKNOWN;
  int fieldType;
  int fieldID;
  [protocol readStructBeginReturningName: NULL];
  while (true) {
    [protocol readFieldBeginReturningName: NULL
              type: &fieldType
              fieldID: &fieldID];
    if (fieldType == TType_STOP) {
      break;
    }
    switch (fieldID) {
    case 1:
      if (fieldType == TType_STRING) {
        reason = [protocol readString];
      } else {
        [TProtocolUtil skipType: fieldType onProtocol: protocol];
      }
      break;
    case 2:
      if (fieldType == TType_I32) {
        type = [protocol readI32];
      } else {
        [TProtocolUtil skipType: fieldType onProtocol: protocol];
      }
      break;
    default:
      [TProtocolUtil skipType: fieldType onProtocol: protocol];
      break;
    }
    [protocol readFieldEnd];
  }
  [protocol readStructEnd];
  return [TApplicationException exceptionWithType: type reason: reason];
}
- (void) write: (id <TProtocol>) protocol
{
  [protocol writeStructBeginWithName: @"TApplicationException"];
  if ([self reason] != nil) {
    [protocol writeFieldBeginWithName: @"message"
                 type: TType_STRING
                 fieldID: 1];
    [protocol writeString: [self reason]];
    [protocol writeFieldEnd];
  }
```

```
  [protocol writeFieldBeginWithName: @"type"
              type: TType_I32
              fieldID: 2];
  [protocol writeI32: mType];
  [protocol writeFieldEnd];
  [protocol writeFieldStop];
  [protocol writeStructEnd];
}
+ (TApplicationException *) exceptionWithType: (int) type
                                       reason: (NSString *) reason
{
  return [[[TApplicationException alloc] initWithType: type
                                               reason: reason] autorelease_stub];
}
@end
#import "_GAMEImageSetter.h"
#import "GAMEImageOperation.h"
#import <libkern/OSAtomic.h>
NSString *const _GAMEImageFadeAnimationKey = @"GAMEImageFade";
const NSTimeInterval _GAMEImageFadeTime = 0.2;
const NSTimeInterval _GAMEImageProgressiveFadeTime = 0.4;
@implementation _GAMEImageSetter {
    dispatch_semaphore_t _lock;
    NSURL *_imageURL;
    NSOperation *_operation;
    int32_t _sentinel;
}
- (instancetype)init {
    self = [super init];
    _lock = dispatch_semaphore_create(1);
    return self;
}
- (NSURL *)imageURL {
    dispatch_semaphore_wait(_lock, DISPATCH_TIME_FOREVER);
    NSURL *imageURL = _imageURL;
    dispatch_semaphore_signal(_lock);
    return imageURL;
}
- (void)dealloc {
    OSAtomicIncrement32(&_sentinel);
    [_operation cancel];
}
- (int32_t)setOperationWithSentinel:(int32_t)sentinel
                                url:(NSURL *)imageURL
                            options:(GAMEImageOptions)options
                            manager:(GAMEImageManager *)manager
                           progress:(GAMEImageProgressBlock)progress
                          transform:(GAMEImageTransformBlock)transform
                         completion:(GAMEImageCompletionBlock)completion {
    if (sentinel != _sentinel) {
```

```
        if (completion) completion(nil, imageURL, GAMEImageFromNone,
GAMEImageStageCancelled, nil);
        return _sentinel;
    }
       NSOperation *operation = [manager requestImageWithURL:imageURL
options:options progress:progress transform:transform completion:completion];
    if (!operation && completion) {
        NSDictionary *userInfo = @{ NSLocalizedDescriptionKey :
@"GAMEImageOperation create failed." };
        completion(nil, imageURL, GAMEImageFromNone, GAMEImageStageFinished,
[NSError errorWithDomain:@"" code:-1 userInfo:userInfo]);
    }
       dispatch_semaphore_wait(_lock, DISPATCH_TIME_FOREVER);
    if (sentinel == _sentinel) {
        if (_operation) [_operation cancel];
        _operation = operation;
        sentinel = OSAtomicIncrement32(&_sentinel);
    } else {
        [operation cancel];
    }
    dispatch_semaphore_signal(_lock);
    return sentinel;
}
- (int32_t)cancel {
    return [self cancelWithNewURL:nil];
}
- (int32_t)cancelWithNewURL:(NSURL *)imageURL {
    int32_t sentinel;
    dispatch_semaphore_wait(_lock, DISPATCH_TIME_FOREVER);
    if (_operation) {
        [_operation cancel];
        _operation = nil;
    }
    _imageURL = imageURL;
    sentinel = OSAtomicIncrement32(&_sentinel);
    dispatch_semaphore_signal(_lock);
    return sentinel;
}
+ (dispatch_queue_t)setterQueue {
    static dispatch_queue_t queue;
    static dispatch_once_t onceToken;
    dispatch_once(&onceToken, ^{
        queue = dispatch_queue_create("", DISPATCH_QUEUE_SERIAL);
        dispatch_set_target_queue(queue,
dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0));
    });
    return queue;
}
@end
#import <TargetConditionals.h>
```

```
#if TARGET_OS_IOS || TARGET_OS_TV#import "AFAutoPurgingImageCache.h"
@interface AFCachedImage : NSObject
@property (nonatomic, strong) UIImage *image;
@property (nonatomic, strong) NSString *identifier;
@property (nonatomic, assign) UInt64 totalBytes;
@property (nonatomic, strong) NSDate *lastAccessDate;
@property (nonatomic, assign) UInt64 currentMemoryUsage;
@end
@implementation AFCachedImage
-(instancetype)initWithImage:(UIImage *)image identifier:(NSString *)identifier
{
    if (self = [self init]) {
        self.image = image;
        self.identifier = identifier;
        CGSize imageSize = CGSizeMake(image.size.width * image.scale,
image.size.height * image.scale);
        CGFloat bytesPerPixel = 4.0;
        CGFloat bytesPerSize = imageSize.width * imageSize.height;
        self.totalBytes = (UInt64)bytesPerPixel * (UInt64)bytesPerSize;
        self.lastAccessDate = [NSDate date];
    }
    return self;
}
- (UIImage*)accessImage {
    self.lastAccessDate = [NSDate date];
    return self.image;
}
- (NSString *)description {
    NSString *descriptionString = [NSString stringWithFormat:@"Idenfitier: %@
lastAccessDate: %@ ", self.identifier, self.lastAccessDate];
    return descriptionString;
}
@end
@interface AFAutoPurgingImageCache ()
@property (nonatomic, strong) NSMutableDictionary <NSString* , AFCachedImage*>
*cachedImages;
@property (nonatomic, assign) UInt64 currentMemoryUsage;
@property (nonatomic, strong) dispatch_queue_t synchronizationQueue;
@end
@implementation AFAutoPurgingImageCache
- (instancetype)init {
    return [self initWithMemoryCapacity:100 * 1024 * 1024
preferredMemoryCapacity:60 * 1024 * 1024];
}
- (instancetype)initWithMemoryCapacity:(UInt64)memoryCapacity
preferredMemoryCapacity:(UInt64)preferredMemoryCapacity {
    if (self = [super init]) {
        self.memoryCapacity = memoryCapacity;
        self.preferredMemoryUsageAfterPurge = preferredMemoryCapacity;
        self.cachedImages = [[NSMutableDictionary alloc] init];
```

```
        NSString *queueName = [NSString stringWithFormat:@"", [[NSUUID UUID]
UUIDString]];
        self.synchronizationQueue = dispatch_queue_create([queueName
cStringUsingEncoding:NSASCIIStringEncoding], DISPATCH_QUEUE_CONCURRENT);
        [[NSNotificationCenter defaultCenter]
         addObserver:self
         selector:@selector(removeAllImages)
         name:UIApplicationDidReceiveMemoryWarningNotification
         object:nil];
    }
    return self;
}
- (void)dealloc {
    [[NSNotificationCenter defaultCenter] removeObserver:self];
}
- (UInt64)memoryUsage {
    __block UInt64 result = 0;
    dispatch_sync(self.synchronizationQueue, ^{
        result = self.currentMemoryUsage;
    });
    return result;
}
- (void)addImage:(UIImage *)image withIdentifier:(NSString *)identifier {
    dispatch_barrier_async(self.synchronizationQueue, ^{
        AFCachedImage *cacheImage = [[AFCachedImage alloc] initWithImage:image
identifier:identifier];
        AFCachedImage *previousCachedImage = self.cachedImages[identifier];
        if (previousCachedImage != nil) {
            self.currentMemoryUsage -= previousCachedImage.totalBytes;
        }
        self.cachedImages[identifier] = cacheImage;
        self.currentMemoryUsage += cacheImage.totalBytes;
    });
    dispatch_barrier_async(self.synchronizationQueue, ^{
        if (self.currentMemoryUsage > self.memoryCapacity) {
            UInt64 bytesToPurge = self.currentMemoryUsage -
self.preferredMemoryUsageAfterPurge;
            NSMutableArray <AFCachedImage*> *sortedImages = [NSMutableArray
arrayWithArray:self.cachedImages.allValues];
            NSSortDescriptor *sortDescriptor = [[NSSortDescriptor alloc]
initWithKey:@"lastAccessDate"

ascending:YES];
            [sortedImages sortUsingDescriptors:@[sortDescriptor]];
            UInt64 bytesPurged = 0;
            for (AFCachedImage *cachedImage in sortedImages) {
                [self.cachedImages removeObjectForKey:cachedImage.identifier];
                bytesPurged += cachedImage.totalBytes;
                if (bytesPurged >= bytesToPurge) {
                    break ;
```

```
                }
            }
            self.currentMemoryUsage -= bytesPurged;
        }
    });
}
- (BOOL)removeImageWithIdentifier:(NSString *)identifier {
    __block BOOL removed = NO;
    dispatch_barrier_sync(self.synchronizationQueue, ^{
        AFCachedImage *cachedImage = self.cachedImages[identifier];
        if (cachedImage != nil) {
            [self.cachedImages removeObjectForKey:identifier];
            self.currentMemoryUsage -= cachedImage.totalBytes;
            removed = YES;
        }
    });
    return removed;
}
- (BOOL)removeAllImages {
    __block BOOL removed = NO;
    dispatch_barrier_sync(self.synchronizationQueue, ^{
        if (self.cachedImages.count > 0) {
            [self.cachedImages removeAllObjects];
            self.currentMemoryUsage = 0;
            removed = YES;
        }
    });
    return removed;
}
- (nullable UIImage *)imageWithIdentifier:(NSString *)identifier {
    __block UIImage *image = nil;
    dispatch_sync(self.synchronizationQueue, ^{
        AFCachedImage *cachedImage = self.cachedImages[identifier];
        image = [cachedImage accessImage];
    });
    return image;
}
- (void)addImage:(UIImage *)image forRequest:(NSURLRequest *)request
withAdditionalIdentifier:(NSString *)identifier {
    [self addImage:image withIdentifier:[self
imageCacheKeyFromURLRequest:request withAdditionalIdentifier:identifier]];
}
- (BOOL)removeImageforRequest:(NSURLRequest *)request
withAdditionalIdentifier:(NSString *)identifier {
    return [self removeImageWithIdentifier:[self
imageCacheKeyFromURLRequest:request withAdditionalIdentifier:identifier]];
}
- (nullable UIImage *)imageforRequest:(NSURLRequest *)request
withAdditionalIdentifier:(NSString *)identifier {
    return [self imageWithIdentifier:[self imageCacheKeyFromURLRequest:request
```

```
withAdditionalIdentifier:identifier]];
}
- (NSString *)imageCacheKeyFromURLRequest:(NSURLRequest *)request
withAdditionalIdentifier:(NSString *)additionalIdentifier {
    NSString *key = request.URL.absoluteString;
    if (additionalIdentifier != nil) {
        key = [key stringByAppendingString:additionalIdentifier];
    }
    return key;
}
- (BOOL)shouldCacheImage:(UIImage *)image forRequest:(NSURLRequest *)request
withAdditionalIdentifier:(nullable NSString *)identifier {
    return YES;
}
@end
#endif
#import "TBinaryProtocol.h"
#import "TProtocolException.h"
#import "TObjective-C.h"
int32_t VERSION_1 = 0x80010000;
int32_t VERSION_MASK = 0xffff0000;
static TBinaryProtocolFactory * gSharedFactory = nil;
@implementation TBinaryProtocolFactory
+ (TBinaryProtocolFactory *) sharedFactory {
  if (gSharedFactory == nil) {
    gSharedFactory = [[TBinaryProtocolFactory alloc] init];
  }
  return gSharedFactory;
}
- (TBinaryProtocol *) newProtocolOnTransport: (id <TTransport>) transport {
  return [[TBinaryProtocol alloc] initWithTransport: transport];
}
@end
@implementation TBinaryProtocol
- (id) initWithTransport: (id <TTransport>) transport
{
  return [self initWithTransport: transport strictRead: NO strictWrite: YES];
}
- (id) initWithTransport: (id <TTransport>) transport
              strictRead: (BOOL) strictRead
             strictWrite: (BOOL) strictWrite
{
  self = [super init];
  mTransport = [transport retain_stub];
  mStrictRead = strictRead;
  mStrictWrite = strictWrite;
  return self;
}
- (int32_t) messageSizeLimit
{
```

```
  return mMessageSizeLimit;
}
- (void) setMessageSizeLimit: (int32_t) sizeLimit
{
  mMessageSizeLimit = sizeLimit;
}
- (void) dealloc
{
  [mTransport release_stub];
  [super dealloc_stub];
}
- (id <TTransport>) transport
{
  return mTransport;
}
- (NSString *) readStringBody: (int) size
{
  char * buffer = malloc(size+1);
  if (!buffer) {
    @throw [TProtocolException exceptionWithName: @"TProtocolException"
                                          reason: [NSString stringWithFormat:
@"Unable to allocate memory in %s, size: %i",
                                                  __PRETTY_FUNCTION__,
                                                  size]];;
  }
  [mTransport readAll: (uint8_t *) buffer offset: 0 length: size];
  buffer[size] = 0;
  NSString * result = [NSString stringWithUTF8String: buffer];
  free(buffer);
  return result;
}
- (void) readMessageBeginReturningName: (NSString **) name
                                  type: (int *) type
                            sequenceID: (int *) sequenceID
{
  int32_t size = [self readI32];
  if (size < 0) {
    int version = size & VERSION_MASK;
    if (version != VERSION_1) {
      @throw [TProtocolException exceptionWithName: @"TProtocolException"
                                            reason: @"Bad version in readMessageBegin"];
    }
    if (type != NULL) {
      *type = size & 0x00FF;
    }
    NSString * messageName = [self readString];
    if (name != NULL) {
      *name = messageName;
    }
    int seqID = [self readI32];
```

```
    if (sequenceID != NULL) {
      *sequenceID = seqID;
    }
  } else {
    if (mStrictRead) {
      @throw [TProtocolException exceptionWithName: @"TProtocolException"
                                            reason: @"Missing version in readMessageBegin,
old client?"];
    }
    if ([self messageSizeLimit] > 0 && size > [self messageSizeLimit]) {
      @throw [TProtocolException exceptionWithName: @"TProtocolException"
                                            reason: [NSString stringWithFormat:
@"Message too big.  Size limit is: %d Message size is: %d",
                                                    mMessageSizeLimit,
                                                    size]];
    }
    NSString * messageName = [self readStringBody: size];
    if (name != NULL) {
      *name = messageName;
    }
    int messageType = [self readByte];
    if (type != NULL) {
      *type = messageType;
    }
    int seqID = [self readI32];
    if (sequenceID != NULL) {
      *sequenceID = seqID;
    }
  }
}
- (void) readMessageEnd {}
- (void) readStructBeginReturningName: (NSString **) name
{
  if (name != NULL) {
    *name = nil;
  }
}
- (void) readStructEnd {}
- (void) readFieldBeginReturningName: (NSString **) name
                               type: (int *) fieldType
                            fieldID: (int *) fieldID
{
  if (name != NULL) {
    *name = nil;
  }
  int ft = [self readByte];
  if (fieldType != NULL) {
    *fieldType = ft;
  }
  if (ft != TType_STOP) {
```

```
    int fid = [self readI16];
    if (fieldID != NULL) {
      *fieldID = fid;
    }
  }
}
- (void) readFieldEnd {}
- (int32_t) readI32
{
  uint8_t i32rd[4];
  [mTransport readAll: i32rd offset: 0 length: 4];
  return
    ((i32rd[0] & 0xff) << 24) |
    ((i32rd[1] & 0xff) << 16) |
    ((i32rd[2] & 0xff) <<  8) |
    ((i32rd[3] & 0xff));
}
- (NSString *) readString
{
  int size = [self readI32];
  return [self readStringBody: size];
}
- (BOOL) readBool
{
  return [self readByte] == 1;
}
- (uint8_t) readByte
{
  uint8_t myByte;
  [mTransport readAll: &myByte offset: 0 length: 1];
  return myByte;
}
- (short) readI16
{
  uint8_t buff[2];
  [mTransport readAll: buff offset: 0 length: 2];
  return (short)
    (((buff[0] & 0xff) << 8) |
     ((buff[1] & 0xff)));
  return 0;
}
- (int64_t) readI64;
{
  uint8_t i64rd[8];
  [mTransport readAll: i64rd offset: 0 length: 8];
  return
    ((int64_t)(i64rd[0] & 0xff) << 56) |
    ((int64_t)(i64rd[1] & 0xff) << 48) |
    ((int64_t)(i64rd[2] & 0xff) << 40) |
    ((int64_t)(i64rd[3] & 0xff) << 32) |
```

```
    ((int64_t)(i64rd[4] & 0xff) << 24) |
    ((int64_t)(i64rd[5] & 0xff) << 16) |
    ((int64_t)(i64rd[6] & 0xff) <<  8) |
    ((int64_t)(i64rd[7] & 0xff));
}
- (double) readDouble;
{
  // FIXME - will this get us into trouble on Power?
  int64_t ieee754 = [self readI64];
  return *((double *) &ieee754);
}
- (NSData *) readBinary
{
  int32_t size = [self readI32];
  uint8_t * buff = malloc(size);
  if (buff == NULL) {
    @throw [TProtocolException
            exceptionWithName: @"TProtocolException"
            reason: [NSString stringWithFormat: @"Out of memory.  Unable to
allocate %d bytes trying to read binary data.",
                              size]];
  }
  [mTransport readAll: buff offset: 0 length: size];
  return [NSData dataWithBytesNoCopy: buff length: size];
}
- (void) readMapBeginReturningKeyType: (int *) keyType
                            valueType: (int *) valueType
                                 size: (int *) size
{
  int kt = [self readByte];
  int vt = [self readByte];
  int s = [self readI32];
  if (keyType != NULL) {
    *keyType = kt;
  }
  if (valueType != NULL) {
    *valueType = vt;
  }
  if (size != NULL) {
    *size = s;
  }
}
- (void) readMapEnd {}
- (void) readSetBeginReturningElementType: (int *) elementType
                                     size: (int *) size
{
  int et = [self readByte];
  int s = [self readI32];
  if (elementType != NULL) {
    *elementType = et;
```

```
  }
  if (size != NULL) {
    *size = s;
  }
}
- (void) readSetEnd {}
- (void) readListBeginReturningElementType: (int *) elementType
                                      size: (int *) size
{
  int et = [self readByte];
  int s = [self readI32];
  if (elementType != NULL) {
    *elementType = et;
  }
  if (size != NULL) {
    *size = s;
  }
}
- (void) readListEnd {}
- (void) writeByte: (uint8_t) value
{
  [mTransport write: &value offset: 0 length: 1];
}
- (void) writeMessageBeginWithName: (NSString *) name
                              type: (int) messageType
                        sequenceID: (int) sequenceID
{
  if (mStrictWrite) {
    int version = VERSION_1 | messageType;
    [self writeI32: version];
    [self writeString: name];
    [self writeI32: sequenceID];
  } else {
    [self writeString: name];
    [self writeByte: messageType];
    [self writeI32: sequenceID];
  }
}
- (void) writeMessageEnd {}
- (void) writeStructBeginWithName: (NSString *) name {}
- (void) writeStructEnd {}
- (void) writeFieldBeginWithName: (NSString *) name
                            type: (int) fieldType
                         fieldID: (int) fieldID
{
  [self writeByte: fieldType];
  [self writeI16: fieldID];
}
  buff[1] = 0xff & value;
  [mTransport write: buff offset: 0 length: 2];
```

```
}
- (void) writeI64: (int64_t) value
{
  uint8_t buff[8];
  buff[0] = 0xFF & (value >> 56);
  buff[1] = 0xFF & (value >> 48);
  buff[2] = 0xFF & (value >> 40);
  buff[3] = 0xFF & (value >> 32);
  buff[4] = 0xFF & (value >> 24);
  buff[5] = 0xFF & (value >> 16);
  buff[6] = 0xFF & (value >> 8);
  buff[7] = 0xFF & value;
  [mTransport write: buff offset: 0 length: 8];
}
- (void) writeDouble: (double) value
{
  // spit out IEEE 754 bits - FIXME - will this get us in trouble on
  // Power?
  [self writeI64: *((int64_t *) &value)];
}
- (void) writeString: (NSString *) value
{
  if (value != nil) {
    const char * utf8Bytes = [value UTF8String];
    size_t length = strlen(utf8Bytes);
    [self writeI32: length];
    [mTransport write: (uint8_t *) utf8Bytes offset: 0 length: length];
  } else {
    // instead of crashing when we get null, let's write out a zero
    // length string
    [self writeI32: 0];
  }
}
- (void) writeBinary: (NSData *) data
{
  [self writeI32: [data length]];
  [mTransport write: [data bytes] offset: 0 length: [data length]];
}
- (void) writeFieldStop
{
  [self writeByte: TType_STOP];
}
- (void) writeFieldEnd {}
- (void) writeMapBeginWithKeyType: (int) keyType
                        valueType: (int) valueType
                             size: (int) size
{
  [self writeByte: keyType];
  [self writeByte: valueType];
  [self writeI32: size];
```

```
}
- (void) writeMapEnd {}
- (void) writeSetBeginWithElementType: (int) elementType
                                 size: (int) size
{
  [self writeByte: elementType];
  [self writeI32: size];
}
- (void) writeSetEnd {}
- (void) writeListBeginWithElementType: (int) elementType
                                  size: (int) size
{
  [self writeByte: elementType];
  [self writeI32: size];
}
- (void) writeListEnd {}
- (void) writeBool: (BOOL) value
{
  [self writeByte: (value ? 1 : 0)];
}
@end
#import "PromiseKit-Swift.h"
#else
    #import <PromiseKit/PromiseKit-Swift.h>
#endif
#import "PMKCallVariadicBlock.m"
#import "AnyPromise+Private.h"
#import "AnyPromise.h"
NSString *const PMKErrorDomain = @"PMKErrorDomain";
@implementation AnyPromise {
    __AnyPromise *d;
}
- (instancetype)initWith__D:(__AnyPromise *)dd {
    self = [super init];
    if (self) self->d = dd;
    return self;
}
- (instancetype)initWithResolver:(PMKResolver __strong *)resolver {
    self = [super init];
    if (self)
        d = [[__AnyPromise alloc] initWithResolver:^(void (^resolve)(id)) {
            *resolver = resolve;
        }];
    return self;
}
+ (instancetype)promiseWithResolverBlock:(void (^)(PMKResolver
_Nonnull))resolveBlock {
    id d = [[__AnyPromise alloc] initWithResolver:resolveBlock];
    return [[self alloc] initWith__D:d];
}
```

```
+ (instancetype)promiseWithValue:(id)value {
    //TODO provide a more efficient route for sealed promises
    id d = [[__AnyPromise alloc] initWithResolver:^(void (^resolve)(id)) {
        resolve(value);
    }];
    return [[self alloc] initWith__D:d];
}
//TODO remove if possible, but used by when.m
- (void)__pipe:(void (^)(id _Nullable))block {
    [d __pipe:block];
}
//NOTE used by AnyPromise.swift
- (id)__d {
    return d;
}
- (AnyPromise *(^)(id))then {
    return ^(id block) {
        return [self->d __thenOn:dispatch_get_main_queue() execute:^(id obj) {
            return PMKCallVariadicBlock(block, obj);
        }];
    };
}
- (AnyPromise *(^)(dispatch_queue_t, id))thenOn {
    return ^(dispatch_queue_t queue, id block) {
        return [self->d __thenOn:queue execute:^(id obj) {
            return PMKCallVariadicBlock(block, obj);
        }];
    };
}
- (AnyPromise *(^)(id))thenInBackground {
    return ^(id block) {
        return [self->d __thenOn:dispatch_get_global_queue(0, 0) execute:^(id
obj) {
            return PMKCallVariadicBlock(block, obj);
        }];
    };
}
- (AnyPromise *(^)(dispatch_queue_t, id))catchOn {
    return ^(dispatch_queue_t q, id block) {
        return [self->d __catchOn:q execute:^(id obj) {
            return PMKCallVariadicBlock(block, obj);
        }];
    };
}
- (AnyPromise *(^)(id))catch {
    return ^(id block) {
        return [self->d __catchOn:dispatch_get_main_queue() execute:^(id obj) {
            return PMKCallVariadicBlock(block, obj);
        }];
    };
```

```
}
- (AnyPromise *(^)(id))catchInBackground {
    return ^(id block) {
        return [self->d __catchOn:dispatch_get_global_queue(0, 0) execute:^(id
obj) {
            return PMKCallVariadicBlock(block, obj);
        }];
    };
}
- (AnyPromise *(^)(dispatch_block_t))ensure {
    return ^(dispatch_block_t block) {
        return [self->d __ensureOn:dispatch_get_main_queue() execute:block];
    };
}
- (AnyPromise *(^)(dispatch_queue_t, dispatch_block_t))ensureOn {
    return ^(dispatch_queue_t queue, dispatch_block_t block) {
        return [self->d __ensureOn:queue execute:block];
    };
}
- (id)wait {
    return [d __wait];
}
- (BOOL)pending {
    return [[d valueForKey:@"__pending"] boolValue];
}
- (BOOL)rejected {
    return IsError([d __value]);
}
- (BOOL)fulfilled {
    return !self.rejected;
}
- (id)value {
    id obj = [d __value];
    if ([obj isKindOfClass:[PMKArray class]]) {
        return obj[0];
    } else {
        return obj;
    }
}
@end
@implementation AnyPromise (Adapters)
+ (instancetype)promiseWithAdapterBlock:(void (^)(PMKAdapter))block {
    return [self promiseWithResolverBlock:^(PMKResolver resolve) {
        block(^(id value, id error){
            resolve(error ?: value);
        });
    }];
}
+ (instancetype)promiseWithIntegerAdapterBlock:(void
(^)(PMKIntegerAdapter))block {
```

```
(^)(PMKIntegerAdapter))block {
    return [self promiseWithResolverBlock:^(PMKResolver resolve) {
        block(^(NSInteger value, id error){
            if (error) {
                resolve(error);
            } else {
                resolve(@(value));
            }
        });
    }];
}
+ (instancetype)promiseWithBooleanAdapterBlock:(void (^)(PMKBooleanAdapter
adapter))block {
    return [self promiseWithResolverBlock:^(PMKResolver resolve) {
        block(^(BOOL value, id error){
            if (error) {
                resolve(error);
            } else {
                resolve(@(value));
            }
        });
    }];
}
@end
#import "Aspects.h"
#import <libkern/OSAtomic.h>
#import <objc/runtime.h>
#import <objc/message.h>
#define AspectLog(...)
//#define AspectLog(...) do { NSLog(__VA_ARGS__); }while(0)
#define AspectLogError(...) do { NSLog(__VA_ARGS__); }while(0)
// Block internals.
typedef NS_OPTIONS(int, AspectBlockFlags) {
    AspectBlockFlagsHasCopyDisposeHelpers = (1 << 25),
    AspectBlockFlagsHasSignature          = (1 << 30)
};
typedef struct _AspectBlock {
    __unused Class isa;
    AspectBlockFlags flags;
    __unused int reserved;
    void (__unused *invoke)(struct _AspectBlock *block, ...);
    struct {
        unsigned long int reserved;
        unsigned long int size;
        unsigned long int size;
        // requires AspectBlockFlagsHasCopyDisposeHelpers
        void (*copy)(void *dst, const void *src);
        void (*dispose)(const void *);
        // requires AspectBlockFlagsHasSignature
        const char *signature;
```

```
        const char *layout;
    } *descriptor;
    // imported variables
} *AspectBlockRef;
@interface AspectInfo : NSObject <AspectInfo>
- (id)initWithInstance:(__unsafe_unretained id)instance
invocation:(NSInvocation *)invocation;
@property (nonatomic, unsafe_unretained, readonly) id instance;
@property (nonatomic, strong, readonly) NSArray *arguments;
@property (nonatomic, strong, readonly) NSInvocation *originalInvocation;
@end
// Tracks a single aspect.
@interface AspectIdentifier : NSObject
+ (instancetype)identifierWithSelector:(SEL)selector object:(id)object
options:(AspectOptions)options block:(id)block error:(NSError **)error;
- (BOOL)invokeWithInfo:(id<AspectInfo>)info;
@property (nonatomic, assign) SEL selector;
@property (nonatomic, strong) id block;
@property (nonatomic, strong) NSMethodSignature *blockSignature;
@property (nonatomic, weak) id object;
@property (nonatomic, assign) AspectOptions options;
@end
// Tracks all aspects for an object/class.
@interface AspectsContainer : NSObject
- (void)addAspect:(AspectIdentifier *)aspect
withOptions:(AspectOptions)injectPosition;
- (BOOL)removeAspect:(id)aspect;
- (BOOL)hasAspects;
@property (atomic, copy) NSArray *beforeAspects;
@property (atomic, copy) NSArray *insteadAspects;
@property (atomic, copy) NSArray *afterAspects;
@end
@interface AspectTracker : NSObject
- (id)initWithTrackedClass:(Class)trackedClass parent:(AspectTracker *)parent;
@property (nonatomic, strong) Class trackedClass;
@property (nonatomic, strong) NSMutableSet *selectorNames;
@property (nonatomic, weak) AspectTracker *parentEntry;
@end
@interface NSInvocation (Aspects)
- (NSArray *)aspects_arguments;
@end
#define AspectPositionFilter 0x07
#define AspectError(errorCode, errorDescription) do { \
AspectLogError(@"Aspects: %@", errorDescription); \
if (error) { *error = [NSError errorWithDomain:AspectErrorDomain code:errorCode
userInfo:@{NSLocalizedDescriptionKey: errorDescription}]; }}while(0)
NSString *const AspectErrorDomain = @"AspectErrorDomain";
static NSString *const AspectsSubclassSuffix = @"_Aspects_";
static NSString *const AspectsMessagePrefix = @"aspects_";
@implementation NSObject (Aspects)
```

```
////////////////////////////////////////////////////////////////////////////////
/////////////
#pragma mark - Public Aspects API
+ (id<AspectToken>)aspect_hookSelector:(SEL)selector
                       withOptions:(AspectOptions)options
                        usingBlock:(id)block
                             error:(NSError **)error {
    return aspect_add((id)self, selector, options, block, error);
}
/// @return A token which allows to later deregister the aspect.
- (id<AspectToken>)aspect_hookSelector:(SEL)selector
                       withOptions:(AspectOptions)options
                        usingBlock:(id)block
                             error:(NSError **)error {
    return aspect_add(self, selector, options, block, error);
}
////////////////////////////////////////////////////////////////////////////////
/////////////
#pragma mark - Private Helper
static id aspect_add(id self, SEL selector, AspectOptions options, id block,
NSError **error) {
    NSCParameterAssert(self);
    NSCParameterAssert(selector);
    NSCParameterAssert(block);
    __block AspectIdentifier *identifier = nil;
    aspect_performLocked(^{
        if (aspect_isSelectorAllowedAndTrack(self, selector, options, error)) {
            AspectsContainer *aspectContainer =
aspect_getContainerForObject(self, selector);
            identifier = [AspectIdentifier identifierWithSelector:selector
object:self options:options block:block error:error];
            if (identifier) {
                [aspectContainer addAspect:identifier withOptions:options];
                // Modify the class to allow message interception.
                aspect_prepareClassAndHookSelector(self, selector, error);
            }
        }
    });
    return identifier;
}
static BOOL aspect_remove(AspectIdentifier *aspect, NSError **error) {
    NSCAssert([aspect isKindOfClass:AspectIdentifier.class], @"Must have correct
type.");
    __block BOOL success = NO;
    aspect_performLocked(^{
        id self = aspect.object; // strongify
        if (self) {
            AspectsContainer *aspectContainer =
aspect_getContainerForObject(self, aspect.selector);
            success = [aspectContainer removeAspect:aspect];
```

```
            aspect_cleanupHookedClassAndSelector(self, aspect.selector);
            // destroy token
            aspect.object = nil;
            aspect.block = nil;
            aspect.selector = NULL;
        }else {
            NSString *errrorDesc = [NSString stringWithFormat:@"Unable to
deregister hook. Object already deallocated: %@", aspect];
            AspectError(AspectErrorRemoveObjectAlreadyDeallocated,
errrorDesc);
        }
    });
    return success;
}
static void aspect_performLocked(dispatch_block_t block) {
    static OSSpinLock aspect_lock = OS_SPINLOCK_INIT;
    OSSpinLockLock(&aspect_lock);
    block();
    OSSpinLockUnlock(&aspect_lock);
}
static SEL aspect_aliasForSelector(SEL selector) {
    NSCParameterAssert(selector);
    return NSSelectorFromString([AspectsMessagePrefix
stringByAppendingFormat:@"_%@", NSStringFromSelector(selector)]);
}
static NSMethodSignature *aspect_blockMethodSignature(id block, NSError **error)
{
    AspectBlockRef layout = (__bridge void *)block;
    if (!(layout->flags & AspectBlockFlagsHasSignature)) {
        NSString *description = [NSString stringWithFormat:@"The block %@ doesn't
contain a type signature.", block];
        AspectError(AspectErrorMissingBlockSignature, description);
        return nil;
    }
    void *desc = layout->descriptor;
    desc += 2 * sizeof(unsigned long int);
    if (layout->flags & AspectBlockFlagsHasCopyDisposeHelpers) {
        desc += 2 * sizeof(void *);
    }
    if (!desc) {
        NSString *description = [NSString stringWithFormat:@"The block %@ doesn't
has a type signature.", block];
        AspectError(AspectErrorMissingBlockSignature, description);
        return nil;
    }
    const char *signature = (*(const char **)desc);
    return [NSMethodSignature signatureWithObjCTypes:signature];
}
static BOOL aspect_isCompatibleBlockSignature(NSMethodSignature *blockSignature,
id object, SEL selector, NSError **error) {
```

```
    NSCParameterAssert(blockSignature);
    NSCParameterAssert(object);
    NSCParameterAssert(selector);
    BOOL signaturesMatch = YES;
    NSMethodSignature *methodSignature = [[object class]
instanceMethodSignatureForSelector:selector];
    if (blockSignature.numberOfArguments > methodSignature.numberOfArguments) {
        signaturesMatch = NO;
    }else {
        if (blockSignature.numberOfArguments > 1) {
            const char *blockType = [blockSignature getArgumentTypeAtIndex:1];
            if (blockType[0] != '@') {
                signaturesMatch = NO;
            }
        }
        // Argument 0 is self/block, argument 1 is SEL or id<AspectInfo>. We start
comparing at argument 2.
        // The block can have less arguments than the method, that's ok.
        if (signaturesMatch) {
            for (NSUInteger idx = 2; idx < blockSignature.numberOfArguments; idx++)
{
                const char *methodType = [methodSignature
getArgumentTypeAtIndex:idx];
                const char *blockType = [blockSignature
getArgumentTypeAtIndex:idx];
                // Only compare parameter, not the optional type data.
                if (!methodType || !blockType || methodType[0] != blockType[0])
{
                    signaturesMatch = NO; break;
                }
            }
        }
    }
    if (!signaturesMatch) {
        NSString *description = [NSString stringWithFormat:@"Blog signature %@
doesn't match %@.", blockSignature, methodSignature];
        AspectError(AspectErrorIncompatibleBlockSignature, description);
        return NO;
    }
    return YES;
}
//////////////////////////////////////////////////////////////////////////////
/////////////
#pragma mark - Class + Selector Preparation
static BOOL aspect_isMsgForwardIMP(IMP impl) {
    return impl == _objc_msgForward
#if !defined(__arm64__)
    || impl == (IMP)_objc_msgForward_stret
#endif
    ;
```

```
}
static IMP aspect_getMsgForwardIMP(NSObject *self, SEL selector) {
    IMP msgForwardIMP = _objc_msgForward;
#if !defined(__arm64__)
    // As an ugly internal runtime implementation detail in the 32bit runtime, we
need to determine of the method we hook returns a struct or anything larger than
id.
    //     Method method = class_getInstanceMethod(self.class, selector);
    const char *encoding = method_getTypeEncoding(method);
    BOOL methodReturnsStructValue = encoding[0] == _C_STRUCT_B;
    if (methodReturnsStructValue) {
        @try {
            NSUInteger valueSize = 0;
            NSGetSizeAndAlignment(encoding, &valueSize, NULL);
            if (valueSize == 1 || valueSize == 2 || valueSize == 4 || valueSize
== 8) {
                methodReturnsStructValue = NO;
            }
        } @catch (NSException *e) {}
    }
    if (methodReturnsStructValue) {
        msgForwardIMP = (IMP)_objc_msgForward_stret;
    }
#endif
    return msgForwardIMP;
}
static void aspect_prepareClassAndHookSelector(NSObject *self, SEL selector,
NSError **error) {
    NSCParameterAssert(selector);
    Class klass = aspect_hookClass(self, error);
    Method targetMethod = class_getInstanceMethod(klass, selector);
    IMP targetMethodIMP = method_getImplementation(targetMethod);
    if (!aspect_isMsgForwardIMP(targetMethodIMP)) {
        // Make a method alias for the existing method implementation, it not
already copied.
        const char *typeEncoding = method_getTypeEncoding(targetMethod);
        SEL aliasSelector = aspect_aliasForSelector(selector);
        if (![klass instancesRespondToSelector:aliasSelector]) {
            __unused BOOL addedAlias = class_addMethod(klass, aliasSelector,
method_getImplementation(targetMethod), typeEncoding);
            NSCAssert(addedAlias, @"Original implementation for %@ is already
copied to %@ on %@", NSStringFromSelector(selector),
NSStringFromSelector(aliasSelector), klass);
        }
        // We use forwardInvocation to hook in.
        class_replaceMethod(klass, selector, aspect_getMsgForwardIMP(self,
selector), typeEncoding);
        AspectLog(@"Aspects: Installed hook for -[%@ %@].", klass,
NSStringFromSelector(selector));
    }
```

```
}
// Will undo the runtime changes made.
static void aspect_cleanupHookedClassAndSelector(NSObject *self, SEL selector) {
    NSCParameterAssert(self);
    NSCParameterAssert(selector);
    Class klass = object_getClass(self);
    BOOL isMetaClass = class_isMetaClass(klass);
    if (isMetaClass) {
        klass = (Class)self;
    }
    // Check if the method is marked as forwarded and undo that.
    Method targetMethod = class_getInstanceMethod(klass, selector);
    IMP targetMethodIMP = method_getImplementation(targetMethod);
    if (aspect_isMsgForwardIMP(targetMethodIMP)) {
        // Restore the original method implementation.
        const char *typeEncoding = method_getTypeEncoding(targetMethod);
        SEL aliasSelector = aspect_aliasForSelector(selector);
        Method originalMethod = class_getInstanceMethod(klass, aliasSelector);
        IMP originalIMP = method_getImplementation(originalMethod);
        NSCAssert(originalMethod, @"Original implementation for %@ not found %@
on %@", NSStringFromSelector(selector), NSStringFromSelector(aliasSelector),
klass);
        class_replaceMethod(klass, selector, originalIMP, typeEncoding);
        AspectLog(@"Aspects: Removed hook for -[%@ %@].", klass,
NSStringFromSelector(selector));
    }
    // Deregister global tracked selector
    aspect_deregisterTrackedSelector(self, selector);
    // Get the aspect container and check if there are any hooks remaining. Clean
up if there are not.
    AspectsContainer *container = aspect_getContainerForObject(self, selector);
    if (!container.hasAspects) {
        // Destroy the container
        aspect_destroyContainerForObject(self, selector);
        // Figure out how the class was modified to undo the changes.
        NSString *className = NSStringFromClass(klass);
        if ([className hasSuffix:AspectsSubclassSuffix]) {
            Class originalClass = NSClassFromString([className
stringByReplacingOccurrencesOfString:AspectsSubclassSuffix withString:@""]);
            NSCAssert(originalClass != nil, @"Original class must exist");
            object_setClass(self, originalClass);
            AspectLog(@"Aspects: %@ has been restored.",
NSStringFromClass(originalClass));
            // We can only dispose the class pair if we can ensure that no instances
exist using our subclass.
            // Since we don't globally track this, we can't ensure this - but there's
also not much overhead in keeping it around.
            //objc_disposeClassPair(object.class);
        }else {
            // Class is most likely swizzled in place. Undo that.
```

```
            if (isMetaClass) {
                aspect_undoSwizzleClassInPlace((Class)self);
            }
        }
    }
}
////////////////////////////////////////////////////////////////////////////////
/////////////
#pragma mark - Hook Class
static Class aspect_hookClass(NSObject *self, NSError **error) {
    NSCParameterAssert(self);
    Class statedClass = self.class;
    Class baseClass = object_getClass(self);
    NSString *className = NSStringFromClass(baseClass);
    // Already subclassed
    if ([className hasSuffix:AspectsSubclassSuffix]) {
        return baseClass;
        // We swizzle a class object, not a single object.
    }else if (class_isMetaClass(baseClass)) {
        return aspect_swizzleClassInPlace((Class)self);
        // Probably a KVO'ed class. Swizzle in place. Also swizzle meta classes
in place.
    }else if (statedClass != baseClass) {
        return aspect_swizzleClassInPlace(baseClass);
    }
    // Default case. Create dynamic subclass.
    const char *subclassName = [className
stringByAppendingString:AspectsSubclassSuffix].UTF8String;
    Class subclass = objc_getClass(subclassName);
    if (subclass == nil) {
        subclass = objc_allocateClassPair(baseClass, subclassName, 0);
        if (subclass == nil) {
            NSString *errrorDesc = [NSString
stringWithFormat:@"objc_allocateClassPair failed to allocate class %s.",
subclassName];
            AspectError(AspectErrorFailedToAllocateClassPair, errrorDesc);
            return nil;
        }
        aspect_swizzleForwardInvocation(subclass);
        aspect_hookedGetClass(subclass, statedClass);
        aspect_hookedGetClass(object_getClass(subclass), statedClass);
        objc_registerClassPair(subclass);
    }
    object_setClass(self, subclass);
    return subclass;
}
static NSString *const AspectsForwardInvocationSelectorName =
@"__aspects_forwardInvocation:";
static void aspect_swizzleForwardInvocation(Class klass) {
    NSCParameterAssert(klass);
```

```
    // If there is no method, replace will act like class_addMethod.
    IMP originalImplementation = class_replaceMethod(klass,
@selector(forwardInvocation:), (IMP)__ASPECTS_ARE_BEING_CALLED__, "v@:@");
    if (originalImplementation) {
        class_addMethod(klass,
NSSelectorFromString(AspectsForwardInvocationSelectorName),
originalImplementation, "v@:@");
    }
    AspectLog(@"Aspects: %@ is now aspect aware.", NSStringFromClass(klass));
}
static void aspect_undoSwizzleForwardInvocation(Class klass) {
    NSCParameterAssert(klass);
    Method originalMethod = class_getInstanceMethod(klass,
NSSelectorFromString(AspectsForwardInvocationSelectorName));
    Method objectMethod = class_getInstanceMethod(NSObject.class,
@selector(forwardInvocation:));
    // There is no class_removeMethod, so the best we can do is to retore the original
implementation, or use a dummy.
    IMP originalImplementation = method_getImplementation(originalMethod ?:
objectMethod);
    class_replaceMethod(klass, @selector(forwardInvocation:),
originalImplementation, "v@:@");
    AspectLog(@"Aspects: %@ has been restored.", NSStringFromClass(klass));
}
static void aspect_hookedGetClass(Class class, Class statedClass) {
    NSCParameterAssert(class);
    NSCParameterAssert(statedClass);
    Method method = class_getInstanceMethod(class, @selector(class));
    IMP newIMP = imp_implementationWithBlock(^(id self) {
        return statedClass;
    });
    class_replaceMethod(class, @selector(class), newIMP,
method_getTypeEncoding(method));
}
////////////////////////////////////////////////////////////////////////////////
/////////////
#pragma mark - Swizzle Class In Place
static void _aspect_modifySwizzledClasses(void (^block)(NSMutableSet
*swizzledClasses)) {
    static NSMutableSet *swizzledClasses;
    static dispatch_once_t pred;
    dispatch_once(&pred, ^{
        swizzledClasses = [NSMutableSet new];
    });
    @synchronized(swizzledClasses) {
        block(swizzledClasses);
    }
}
static Class aspect_swizzleClassInPlace(Class klass) {
    NSCParameterAssert(klass);
```

```
    NSString *className = NSStringFromClass(klass);
    _aspect_modifySwizzledClasses(^(NSMutableSet *swizzledClasses) {
        if (![swizzledClasses containsObject:className]) {
            aspect_swizzleForwardInvocation(klass);
            [swizzledClasses addObject:className];
        }
    });
    return klass;
}
static void aspect_undoSwizzleClassInPlace(Class klass) {
    NSCParameterAssert(klass);
    NSString *className = NSStringFromClass(klass);
    _aspect_modifySwizzledClasses(^(NSMutableSet *swizzledClasses) {
        if ([swizzledClasses containsObject:className]) {
            aspect_undoSwizzleForwardInvocation(klass);
            [swizzledClasses removeObject:className];
        }
    });
}
////////////////////////////////////////////////////////////////////////////////
/////////////
#pragma mark - Aspect Invoke Point
// This is a macro so we get a cleaner stack trace.
#define aspect_invoke(aspects, info) \
for (AspectIdentifier *aspect in aspects) {\
    [aspect invokeWithInfo:info];\
    if (aspect.options & AspectOptionAutomaticRemoval) { \
        aspectsToRemove = [aspectsToRemove?:@[] arrayByAddingObject:aspect]; \
    } \
}
// This is the swizzled forwardInvocation: method.
static void __ASPECTS_ARE_BEING_CALLED__(__unsafe_unretained NSObject *self, SEL
selector, NSInvocation *invocation) {
    NSCParameterAssert(self);
    NSCParameterAssert(invocation);
    SEL originalSelector = invocation.selector;
    SEL aliasSelector = aspect_aliasForSelector(invocation.selector);
    invocation.selector = aliasSelector;
    AspectsContainer *objectContainer = objc_getAssociatedObject(self,
aliasSelector);
    AspectsContainer *classContainer =
aspect_getContainerForClass(object_getClass(self), aliasSelector);
    AspectInfo *info = [[AspectInfo alloc] initWithInstance:self
invocation:invocation];
    NSArray *aspectsToRemove = nil;
    // Before hooks.
    aspect_invoke(classContainer.beforeAspects, info);
    aspect_invoke(objectContainer.beforeAspects, info);
    // Instead hooks.
    BOOL respondsToAlias = YES;
```

```
    if (objectContainer.insteadAspects.count ||
classContainer.insteadAspects.count) {
        aspect_invoke(classContainer.insteadAspects, info);
        aspect_invoke(objectContainer.insteadAspects, info);
    }else {
        Class klass = object_getClass(invocation.target);
        do {
            if ((respondsToAlias = [klass
instancesRespondToSelector:aliasSelector])) {
                [invocation invoke];
                break;
            }
        }while (!respondsToAlias && (klass = class_getSuperclass(klass)));
    }
    // After hooks.
    aspect_invoke(classContainer.afterAspects, info);
    aspect_invoke(objectContainer.afterAspects, info);
    // If no hooks are installed, call original implementation (usually to throw
an exception)
    if (!respondsToAlias) {
        invocation.selector = originalSelector;
        SEL originalForwardInvocationSEL =
NSSelectorFromString(AspectsForwardInvocationSelectorName);
        if ([self respondsToSelector:originalForwardInvocationSEL]) {
            ((void( *)(id, SEL, NSInvocation *))objc_msgSend)(self,
originalForwardInvocationSEL, invocation);
        }else {
            [self doesNotRecognizeSelector:invocation.selector];
        }
    }
    // Remove any hooks that are queued for deregistration.
    [aspectsToRemove makeObjectsPerformSelector:@selector(remove)];
}
#undef aspect_invoke
////////////////////////////////////////////////////////////////////////////////
/////////////
#pragma mark - Aspect Container Management
// Loads or creates the aspect container.
static AspectsContainer *aspect_getContainerForObject(NSObject *self, SEL
selector) {
    NSCParameterAssert(self);
    SEL aliasSelector = aspect_aliasForSelector(selector);
    AspectsContainer *aspectContainer = objc_getAssociatedObject(self,
aliasSelector);
    if (!aspectContainer) {
        aspectContainer = [AspectsContainer new];
        objc_setAssociatedObject(self, aliasSelector, aspectContainer,
OBJC_ASSOCIATION_RETAIN);
    }
    return aspectContainer;
```

```
}
static AspectsContainer *aspect_getContainerForClass(Class klass, SEL selector)
{
    NSCParameterAssert(klass);
    AspectsContainer *classContainer = nil;
    do {
        classContainer = objc_getAssociatedObject(klass, selector);
        if (classContainer.hasAspects) break;
    }while ((klass = class_getSuperclass(klass)));
    return classContainer;
}
static void aspect_destroyContainerForObject(id<NSObject> self, SEL selector) {
    NSCParameterAssert(self);
    SEL aliasSelector = aspect_aliasForSelector(selector);
    objc_setAssociatedObject(self, aliasSelector, nil,
OBJC_ASSOCIATION_RETAIN);
}
////////////////////////////////////////////////////////////////////////////////
/////////////
#pragma mark - Selector Blacklist Checking
static NSMutableDictionary *aspect_getSwizzledClassesDict() {
    static NSMutableDictionary *swizzledClassesDict;
    static dispatch_once_t pred;
    dispatch_once(&pred, ^{
        swizzledClassesDict = [NSMutableDictionary new];
    });
    return swizzledClassesDict;
}
static BOOL aspect_isSelectorAllowedAndTrack(NSObject *self, SEL selector,
AspectOptions options, NSError **error) {
    static NSSet *disallowedSelectorList;
    static dispatch_once_t pred;
    dispatch_once(&pred, ^{
        disallowedSelectorList = [NSSet setWithObjects:@"retain", @"release",
@"autorelease", @"forwardInvocation:", nil];
    });
    // Check against the blacklist.
    NSString *selectorName = NSStringFromSelector(selector);
    if ([disallowedSelectorList containsObject:selectorName]) {
        NSString *errorDescription = [NSString stringWithFormat:@"Selector %@ is
blacklisted.", selectorName];
        AspectError(AspectErrorSelectorBlacklisted, errorDescription);
        return NO;
    }
    // Additional checks.
    AspectOptions position = options&AspectPositionFilter;
    if ([selectorName isEqualToString:@"dealloc"] && position !=
AspectPositionBefore) {
        NSString *errorDesc = @"AspectPositionBefore is the only valid position
when hooking dealloc.";
```

```
        AspectError(AspectErrorSelectorDeallocPosition, errorDesc);
        return NO;
    }
    if (![self respondsToSelector:selector] && ![self.class
instancesRespondToSelector:selector]) {
        NSString *errorDesc = [NSString stringWithFormat:@"Unable to find
selector -[%@ %@].", NSStringFromClass(self.class), selectorName];
        AspectError(AspectErrorDoesNotRespondToSelector, errorDesc);
        return NO;
    }
    // Search for the current class and the class hierarchy IF we are modifying
a class object
    if (class_isMetaClass(object_getClass(self))) {
        Class klass = [self class];
        NSMutableDictionary *swizzledClassesDict =
aspect_getSwizzledClassesDict();
        Class currentClass = [self class];
        do {
            AspectTracker *tracker = swizzledClassesDict[currentClass];
            if ([tracker.selectorNames containsObject:selectorName]) {
                // Find the topmost class for the log.
                if (tracker.parentEntry) {
                    AspectTracker *topmostEntry = tracker.parentEntry;
                    while (topmostEntry.parentEntry) {
                        topmostEntry = topmostEntry.parentEntry;
                    }
                    NSString *errorDescription = [NSString
stringWithFormat:@"Error: %@ already hooked in %@. A method can only be hooked once
per class hierarchy.", selectorName,
NSStringFromClass(topmostEntry.trackedClass)];

AspectError(AspectErrorSelectorAlreadyHookedInClassHierarchy,
errorDescription);
                    return NO;
                }else if (klass == currentClass) {
                    // Already modified and topmost!
                    return YES;
                }
            }
        }while ((currentClass = class_getSuperclass(currentClass)));
        // Add the selector as being modified.
        currentClass = klass;
        AspectTracker *parentTracker = nil;
        do {
            AspectTracker *tracker = swizzledClassesDict[currentClass];
            if (!tracker) {
                tracker = [[AspectTracker alloc]
initWithTrackedClass:currentClass parent:parentTracker];
                swizzledClassesDict[(id<NSCopying>)currentClass] = tracker;
            }
```

```
            [tracker.selectorNames addObject:selectorName];
            // All superclasses get marked as having a subclass that is modified.
            parentTracker = tracker;
        }while ((currentClass = class_getSuperclass(currentClass)));
    }
    return YES;
}
static void aspect_deregisterTrackedSelector(id self, SEL selector) {
    if (!class_isMetaClass(object_getClass(self))) return;
    NSMutableDictionary *swizzledClassesDict = aspect_getSwizzledClassesDict();
    NSString *selectorName = NSStringFromSelector(selector);
    Class currentClass = [self class];
    do {
        AspectTracker *tracker = swizzledClassesDict[currentClass];
        if (tracker) {
            [tracker.selectorNames removeObject:selectorName];
            if (tracker.selectorNames.count == 0) {
                [swizzledClassesDict removeObjectForKey:tracker];
            }
        }
    }while ((currentClass = class_getSuperclass(currentClass)));
}
@end
@implementation AspectTracker
- (id)initWithTrackedClass:(Class)trackedClass parent:(AspectTracker *)parent {
    if (self = [super init]) {
        _trackedClass = trackedClass;
        _parentEntry = parent;
        _selectorNames = [NSMutableSet new];
    }
    return self;
}
- (NSString *)description {
    return [NSString stringWithFormat:@"<%@: %@, trackedClass: %@,
selectorNames:%@, parent:%p>", self.class, self,
NSStringFromClass(self.trackedClass), self.selectorNames, self.parentEntry];
}
@end
////////////////////////////////////////////////////////////////////////////////
/////////////
#pragma mark - NSInvocation (Aspects)
@implementation NSInvocation (Aspects)
// Thanks to the ReactiveCocoa team for providing a generic solution for this.
- (id)aspect_argumentAtIndex:(NSUInteger)index {
    const char *argType = [self.methodSignature getArgumentTypeAtIndex:index];
    // Skip const type qualifier.
    if (argType[0] == _C_CONST) argType++;
#define WRAP_AND_RETURN(type) do { type val = 0; [self getArgument:&val
atIndex:(NSInteger)index]; return @(val); } while (0)
    if (strcmp(argType, @encode(id)) == 0 || strcmp(argType, @encode(Class)) ==
```

```
0) {
        __autoreleasing id returnObj;
        [self getArgument:&returnObj atIndex:(NSInteger)index];
        return returnObj;
    } else if (strcmp(argType, @encode(SEL)) == 0) {
        SEL selector = 0;
        [self getArgument:&selector atIndex:(NSInteger)index];
        return NSStringFromSelector(selector);
    } else if (strcmp(argType, @encode(Class)) == 0) {
        __autoreleasing Class theClass = Nil;
        [self getArgument:&theClass atIndex:(NSInteger)index];
        return theClass;
        // Using this list will box the number with the appropriate constructor,
instead of the generic NSValue.
    } else if (strcmp(argType, @encode(char)) == 0) {
        WRAP_AND_RETURN(char);
    } else if (strcmp(argType, @encode(int)) == 0) {
        WRAP_AND_RETURN(int);
    } else if (strcmp(argType, @encode(short)) == 0) {
        WRAP_AND_RETURN(short);
    } else if (strcmp(argType, @encode(long)) == 0) {
        WRAP_AND_RETURN(long);
    } else if (strcmp(argType, @encode(long long)) == 0) {
        WRAP_AND_RETURN(long long);
    } else if (strcmp(argType, @encode(unsigned char)) == 0) {
        WRAP_AND_RETURN(unsigned char);
    } else if (strcmp(argType, @encode(unsigned int)) == 0) {
        WRAP_AND_RETURN(unsigned int);
    } else if (strcmp(argType, @encode(unsigned short)) == 0) {
        WRAP_AND_RETURN(unsigned short);
    } else if (strcmp(argType, @encode(unsigned long)) == 0) {
        WRAP_AND_RETURN(unsigned long);
    } else if (strcmp(argType, @encode(unsigned long long)) == 0) {
        WRAP_AND_RETURN(unsigned long long);
    } else if (strcmp(argType, @encode(float)) == 0) {
        WRAP_AND_RETURN(float);
    } else if (strcmp(argType, @encode(double)) == 0) {
        WRAP_AND_RETURN(double);
    } else if (strcmp(argType, @encode(BOOL)) == 0) {
        WRAP_AND_RETURN(BOOL);
    } else if (strcmp(argType, @encode(bool)) == 0) {
        WRAP_AND_RETURN(BOOL);
    } else if (strcmp(argType, @encode(char *)) == 0) {
        WRAP_AND_RETURN(const char *);
    } else if (strcmp(argType, @encode(void (^)(void))) == 0) {
        __unsafe_unretained id block = nil;
        [self getArgument:&block atIndex:(NSInteger)index];
        return [block copy];
    } else {
        NSUInteger valueSize = 0;
```

```
        NSGetSizeAndAlignment(argType, &valueSize, NULL);
        unsigned char valueBytes[valueSize];
        [self getArgument:valueBytes atIndex:(NSInteger)index];
        return [NSValue valueWithBytes:valueBytes objCType:argType];
    }
    return nil;
#undef WRAP_AND_RETURN
}
- (NSArray *)aspects_arguments {
    NSMutableArray *argumentsArray = [NSMutableArray array];
    for (NSUInteger idx = 2; idx < self.methodSignature.numberOfArguments; idx++)
{
        [argumentsArray addObject:[self aspect_argumentAtIndex:idx] ?:
NSNull.null];
    }
    return [argumentsArray copy];
}
@end
////////////////////////////////////////////////////////////////////////////////
/////////////
#pragma mark - AspectIdentifier
@implementation AspectIdentifier
+ (instancetype)identifierWithSelector:(SEL)selector object:(id)object
options:(AspectOptions)options block:(id)block error:(NSError **)error {
    NSCParameterAssert(block);
    NSCParameterAssert(selector);
    NSMethodSignature *blockSignature = aspect_blockMethodSignature(block,
    NSNumber *v = [self valueForKeyPath:@"transform.scale"];
    return v.doubleValue;
}
- (void)setTransformScale:(CGFloat)v {
    [self setValue:@(v) forKeyPath:@"transform.scale"];
}
- (CGFloat)transformTranslationX {
    NSNumber *v = [self valueForKeyPath:@"transform.translation.x"];
    return v.doubleValue;
}
- (void)setTransformTranslationX:(CGFloat)v {
    [self setValue:@(v) forKeyPath:@"transform.translation.x"];
}
- (CGFloat)transformTranslationY {
    NSNumber *v = [self valueForKeyPath:@"transform.translation.y"];
    return v.doubleValue;
}
- (void)setTransformTranslationY:(CGFloat)v {
    [self setValue:@(v) forKeyPath:@"transform.translation.y"];
}
- (CGFloat)transformTranslationZ {
    NSNumber *v = [self valueForKeyPath:@"transform.translation.z"];
    return v.doubleValue;
```

```
}
- (void)setTransformTranslationZ:(CGFloat)v {
    [self setValue:@(v) forKeyPath:@"transform.translation.z"];
}
- (CGFloat)transformDepth {
    return self.transform.m34;
}
- (void)setTransformDepth:(CGFloat)v {
    CATransform3D d = self.transform;
    d.m34 = v;
    self.transform = d;
}
- (UIViewContentMode)contentMode {
    return YYCAGravityToUIViewContentMode(self.contentsGravity);
}
- (void)setContentMode:(UIViewContentMode)contentMode {
    self.contentsGravity = YYUIViewContentModeToCAGravity(contentMode);
}
- (void)addFadeAnimationWithDuration:(NSTimeInterval)duration
curve:(UIViewAnimationCurve)curve {
    if (duration <= 0) return;
       NSString *mediaFunction;
    switch (curve) {
        case UIViewAnimationCurveEaseInOut: {
            mediaFunction = kCAMediaTimingFunctionEaseInEaseOut;
        } break;
        case UIViewAnimationCurveEaseIn: {
            mediaFunction = kCAMediaTimingFunctionEaseIn;
        } break;
        case UIViewAnimationCurveEaseOut: {
            mediaFunction = kCAMediaTimingFunctionEaseOut;
        } break;
        case UIViewAnimationCurveLinear: {
            mediaFunction = kCAMediaTimingFunctionLinear;
        } break;
        default: {
            mediaFunction = kCAMediaTimingFunctionLinear;
        } break;
    }
       CATransition *transition = [CATransition animation];
    transition.duration = duration;
    transition.timingFunction = [CAMediaTimingFunction
functionWithName:mediaFunction];
    transition.type = kCATransitionFade;
    [self addAnimation:transition forKey:@"yykit.fade"];
}
- (void)removePreviousFadeAnimation {
    [self removeAnimationForKey:@"yykit.fade"];
}
@end
```