Chapter 18

Network

การเขียนโปรแกรมที่เกี่ยวข้องกับระบบเครือข่ายด้วย Objective-C นั้น มีไลบรารีและเฟรมเวิร์คให้ใช้งานแบ่งออกเป็นหลายระดับด้วยกัน โดยเริ่มตั้งแต่ระดับล่างสุดที่ต้องจัดการ network packet เอง ไปจนถึงระดับสูง ในบทนี้จะได้เรียนรู้เกี่ยวกับคลาสและเฟรมเวิร์คต่างๆที่เกี่ยวระบบเครือข่าย รวมไปถึงสถาปัตยกรรมของคลาสที่ใช้งาน เนื้อหาในบทนี้จะครอบคลุมพื้นฐานการใช้งานทั่วไป เช่นการ download , upload ส่วนการใช้งานระดับสูงเช่นการ authentication , cache , cookie และการใช้ API ระดับล่างๆเช่น CFNetwork ไม่ได้คลอบคลุมในบทนี้ ถึงแม้ว่าเนื้อหาในบทนี้ผู้ที่ไม่มีความรู้ทาง network หรือไม่เคยเขียนโปรแกรมที่เกี่ยวกับเครือข่ายมาก่อนสามารถอ่านทำความเข้าใจได้ไม่ยาก แต่ถ้าหากผู้อ่านควรมีความรู้เบื้องต้นเกี่ยวกับโพรโทคอล TCP/IP จะช่วยให้เข้าใจปัญหาและสามารถแก้ไขสิ่งต่างๆที่เกิดขึ้นเมื่อมีข้อผิดพลาดในการเขียนโปรแกรม เพราะเป็นโพรโทคอลหลักที่ใช้ใน cocoa network framework

Foundation Class Method

หากย้อนกลับไปบทก่อนๆ จะพบว่าโปรแกรมที่เราได้เขียนไปนั้นมีการใช้งาน network มาบ้างแล้ว นั่นก็คือการใช้เมธอด ของ Foundation Class ที่มีชื่อ initWithContentsOfURL: หรือมีชื่อลักษณะเดียวกัน ดังเช่นตัวอย่างต่อไปนี้

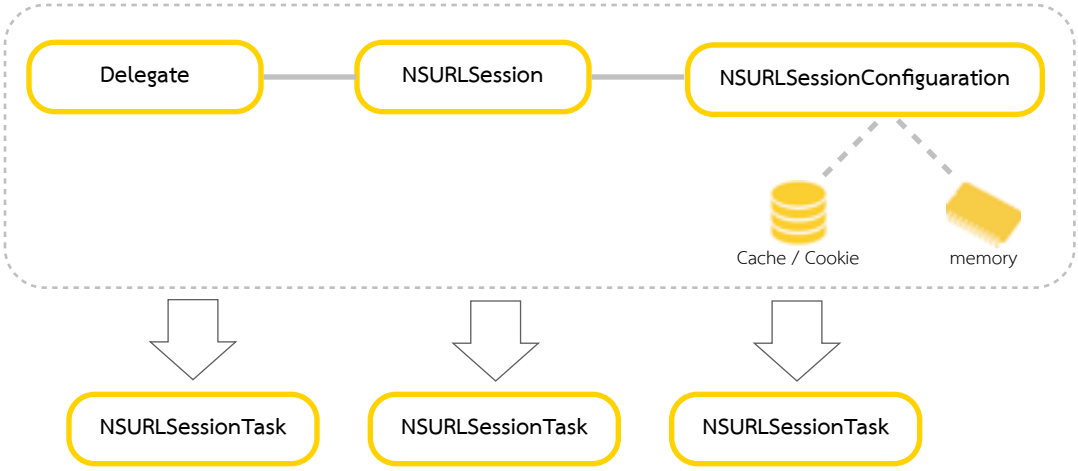
```
NSURL * jsonDict = [NSURL URLWithString: @"http://www.macfeteria.com/example.json"];
NSDictionary * dict = [[NSDictionary alloc] initWithContentsOfURL:jsonDict];
```

การใช้เมธอดที่มากับ Foundation Class มีข้อดีคือ ง่าย ไม่ซับซ้อน และสะดวกมาก แต่เมธอดเหล่านี้ไม่ได้ออกแบบมาเพื่อใช้กับข้อมูลขนาดใหญ่ เช่น ถ้าหากต้องการจะโหลดไฟล์เอกสารขนาด 2 mb ไว้ใน NSString ด้วยการเรียกเมธอด initWithContentsOfURL:encoding:error: คงไม่ใช่เรื่องที่ดีแน่นอน เพราะเมธอดเหล่านี้มีการทำงานแบบ synchronous หรือต้องรอให้เมธอดเหล่านี้ทำงานเสร็จสิ้นเสียก่อนจึงจะสามารถทำอย่างอื่นได้ และในทางตรงกันข้าม ลักษณะการทำงานแบบไม่ต้องรอให้เสร็จจะเรียกว่า asynchronous ถ้าหากเขียนโปรแกรมแบบ console ดังที่เราเขียนๆมาตั้งแต่เริ่มต้นของหนังสือ การรอให้เมธอดที่เกี่ยวกับระบบเครือข่ายทำงานเสร็จสิ้นเสียก่อนคงไม่ใช่ปัญหาใหญ่ แต่เมื่อเขียนโปรแกรมที่มี User Interface เช่น iOS และ Mac OSX จะกลายเป็นเรื่องใหญ่ทันที เพราะ UI ต่างๆจะไม่ตอบสนอง เกิดการหยุดชะงักหรือ freezing เนื่องจากต้องรอให้เมธอดเหล่านี้ทำงานเสร็จเสียก่อนนั่นเอง

NSURLSession

ตั้งแต่แอปเปิ้ลได้ปล่อย Mac OSX 10.9 Maverick และ iOS 7 ก็ได้เพิ่มคลาส NSURLSession เข้ามาเป็นส่วนหนึ่งของ Network Framework ซึ่งเป้าหมายของคลาสนี้คือมาทำหน้าที่แทนคลาส NSURLConnection ที่เคยใช้กันมานาน ดังนั้นเราจึง

ควรใช้คลาสใหม่นี้แทน NSURLConnection และคลาส NSURLSession นี้มีก็ความสามารถที่เหนือกว่า NSURLConnection หลายๆอย่าง เช่นการทำงานในโหมด background หรือความสามารถในการหยุดทำงานชั่วคราว และสั่งให้เริ่มต้นใหม่ หรือทำงานต่อจากสิ่งที่ค้างไว้ได้ เป็นต้น และนอกจากคลาส NSURLSession แล้วยังมีคลาสที่เกี่ยวข้องกันอีกหลายคลาส ดังนั้นก่อนที่จะเริ่มลงมือเขียนโค้ดเราควรเข้าใจโครงสร้างรวมไปถึงส่วนประกอบอื่นๆที่เกี่ยวข้องกันเสียก่อน



จากรูปการทำงานของ NSURLSession API นั้นจะประกอบไปด้วยเซสชั่น (session) และในโปรแกรมก็มีได้หลายๆเซสชั่น ซึ่งแต่ละ session จะประกอบไปด้วยกลุ่มของการทำงานย่อย (task) เช่นการดาวน์โหลด หรือการส่งข้อมูล นอกจาก task แล้วเซสชั่นจะทำงานร่วมกับ delegate และคลาสสำหรับกำหนดค่าต่างๆของเซสชั่น ถ้าจะเปรียบการทำงานของเซสชั่นให้ง่ายขึ้นให้นึกถึงการทำงานของเว็บบราวเซอร์ที่มีหลายหน้าต่าง โดยหน้าต่างแต่ละอันนั้นก็เปรียบเสมือน session ที่ได้กำหนด URL ที่จะใช้งาน และในแต่ละหน้าต่างก็มีการทำงานย่อยหลายๆอย่าง เช่น โหลดรูปภาพ โหลดสคริป เป็นต้น และเมื่อทำลายเซสชั่นลง การทำงานย่อยๆในเซสชั่นนั้นก็จะปิดตัวลง เช่นกัน

NSURLSessionConfiguration

คลาสนี้อาจจะเป็นคลาสแรกที่จะต้องประกาศเมื่อเริ่มใช้งาน NSURLSession API หน้าที่หลักของคลาสนี้คือช่วยให้ผู้ใช้ได้ปรับค่าใช้งานต่างๆก่อนเริ่มเซสชั่นเช่น cache , http header , cookie , security และอื่นๆ คลาสนี้จะถูกเรียกใช้เพียงครั้งเดียวตอนเริ่มต้น และหลักจากที่เซสชั่นได้ทำงานไปแล้ว การปรับเปลี่ยนค่า NSURLSessionConfiguration จะไม่มีผลกระทบอะไรต่อเซสชั่นนั้น การประกาศ NSURLSessionConfiguration จะใช้คลาสเมธอดในการประกาศ ซึ่งสามารถทำได้ 3 เมธอดด้วยกันคือ

+ defaultSessionConfiguration เป็นค่าเริ่มต้นของเซสชั่น ซึ่งจะมีการเก็บ cookie , cache ไว้ในฮาร์ดดิส การทำงานโดยทั่วไปจะใช้เมธอดนี้
+ ephemeralSessionConfiguration จะไม่มีการเก็บข้อมูลไว้ฮาร์ดดิส แต่จะเก็บไว้ที่หน่วยความจำแทน ดังนั้นหากโปรแกรมยกเลิกการใช้เซสชั่น ข้อมูลต่างๆก็จะหายไปด้วย ซึ่งเหมาะกับการทำงานลักษณะ private mode ที่ไม่ต้องการเก็บข้อมูลไว้หลังจากการใช้งาน
+ backgroundSessionConfiguration ถ้าต้องการให้ network ทำงานเมื่อ application หยุดการทำงานชั่วขณะ (suspend) เช่นในกรณีที่ของการทำงานแบบ background mode ของ iOS Application ก็จะใช้การประกาศแบบนี้ อย่างไรก็ตามการ

ทำงานในลักษณะ background นี้มีข้อจำกัดหลายอย่าง เช่น ต้องเป็นโพรโทคอล http หรือ https เท่านั้น , การทำงานต้องเป็น download , upload เท่านั้นไม่สามารถใช้ data task ได้ (จะอธิบาย task แบบ data ภายหลัง) และจะต้องมี delegate เสมอ

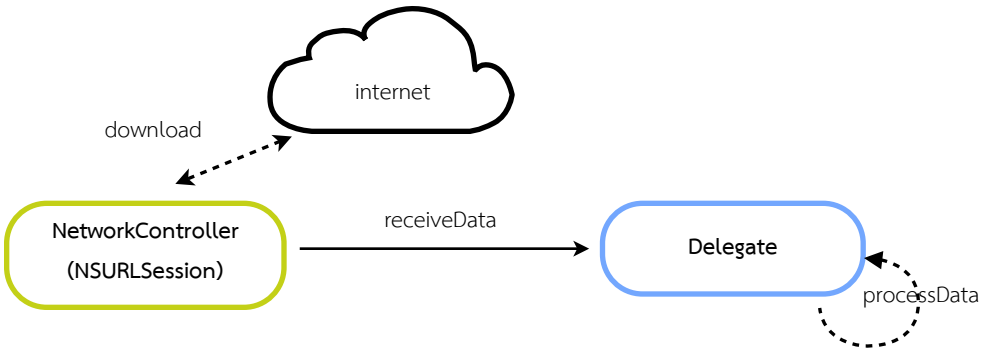
Task

อย่างที่ได้กล่าวไปก่อนหน้านี้ว่า session ประกอบได้ด้วยการทำงานย่อยหรือ task และ NSURLSession ได้แบ่ง task ออกเป็นทั้งหมด 3 แบบด้วยกันคือ

NSURLSessionDownloadTask ใช้สำหรับการ download และจัดเก็บข้อมูลลงไฟล์ ทำงานแบบ background ได้
NSURLSessionUploadDataTask ใช้สำหรับการ upload ไฟล์ ทำงานแบบ background ได้เช่นเดียวกัน
NSURLSessionDataTask จะใช้ส่งและรับข้อมูลด้วย NSData การใช้งานโดยทั่วไปจะประกาศใช้ task แบบนี้ แต่เนื่องจากคลาสนี้มีการรับและส่งข้อมูลด้วย NSData ซึ่งข้อมูลจะเก็บไว้ยังหน่วยความจำ ดังนั้นจึงไม่สามารถใช้ task นี้กับการทำงานแบบ background ได้ ดังนั้นในกรณีที่ต้องการจะให้ระบบ network ยังทำงานในโหมด background ต้องใช้คลาส NSURLSessionDownloadTask หรือ NSURLSessionUploadDataTask

NSURLSession with Delegate

วัฏจักรการทำงานของ NSURLSession จะมีด้วยกันสองแบบ โดยแบบแรกจะใช้ delegate ส่วนแบบที่สองคือไม่ได้ใช้ delegate และโปรแกรมที่จะได้เขียนต่อไปนี้จะเป็นการใช้งานแบบกำหนด delegate ให้กับ session ถ้ายังจำกันได้ในบทที่ 10 เราได้เขียนโปรแกรมจำลองการใช้งาน network โดยมี delegate กันไปแล้ว โดยกระบวนในการทำงาน ก็ได้แสดงดังที่เห็นในรูป



การทำงานของโปรแกรมที่จะได้เขียน ก็มีลักษณะการทำงานเช่นเดียวกัน เพียงแค่เปลี่ยนจาก NetworkController ไปใช้คลาส NSURLSession แทน จากรูปจะเห็นว่าคลาส NetworkController เป็นส่วนที่ใช้ในการติดต่อเครือข่าย ซึ่งจะทำหน้าที่รับหรือส่งข้อมูลอย่างเดียวเท่านั้น แต่ในการจัดการข้อมูลจะมีคลาส MyDelegate มาทำหน้าที่แทน เราจะเขียนคลาส MyDelegate ขึ้นมาใหม่ เพื่อให้รองรับ NSURLSessionDataDelegate โดยมีโค้ดดังนี้

Program 18.1

MyDelegate.h

```
1  #import <Foundation/Foundation.h>
2
3  @interface MyDelegate : NSObject <NSURLSessionDataDelegate>
```

```
@property NSMutableData* netData;
@end
```

MyDelegate.m

```
@implementation MyDelegate
-(id) init
{
    self = [super init];
    if( self != nil)
    {
        _netData = [[NSMutableData alloc] init];
    }
    return self;
}
- (void)URLSession:(NSURLSession *)session
          dataTask:(NSURLSessionDataTask *)dataTask didReceiveData:(NSData *)data
{
    [_netData appendData:data];
}
- (void)URLSession:(NSURLSession *)session
              task:(NSURLSessionTask *)task didCompleteWithError:(NSError *)error
{
    if( error != nil)
        NSLog(@"Error: %@", error.description);
}
@end
```

คลาส MyDelegate ที่ได้เขียนไปมี delegate method ที่ต้องเขียนด้วยกันทั้งหมด 2 เมธอดด้วยกัน โดยเมธอดแรกนั้นจะถูกเรียกเมื่อได้รับข้อมูล และจากโค้ดก็ได้เก็บข้อมูลที่ได้มาไว้ใน netData ส่วนเมธอดต่อมาจะถูกเรียกหลังจาก task ได้ทำงานเสร็จสิ้นลง

main.m

```
#import <Foundation/Foundation.h>
#import "MyDelegate.h"

int main(int argc, const char * argv[])
{

    @autoreleasepool {

        NSString* readme = @"https://dl.dropboxusercontent.com/u/7585756/smith.json";
        NSURL* url = [NSURL URLWithString:readme];

        MyDelegate* delegate = [[MyDelegate alloc] init];

        NSURLSessionConfiguration* config = [NSURLSessionConfiguration
                                             defaultSessionConfiguration];

        NSURLSession *session = [NSURLSession sessionWithConfiguration:config
                                                              delegate:delegate
                                                         delegateQueue:nil];

        NSURLSessionDataTask* task = [session dataTaskWithURL:url];
        [task resume];

        NSDate *start = [NSDate date];

        // Wait until the task is finished
        while (task.state == NSURLSessionTaskStateRunning)
        {
            NSTimeInterval timeInterval = [start timeIntervalSinceNow];
            if( fabs(timeInterval) > 15)
            {
                NSLog(@"Server response time is too long");
                [task cancel];
            }
```

```
        }

        if( [delegate.netData length] > 0)
        {
            NSString* text = [[NSString alloc] initWithData:delegate.netData
                                                   encoding:NSUTF8StringEncoding];

            NSLog(@"%@", text);
        }

    }
    return 0;
}
```

มาถึงในส่วนของโค้ดของโปรแกรมหลัก ได้เริ่มต้นด้วยการประกาศ defaultConfiguration ที่เป็นอ็อบเจ็กของคลาส NSURLSessionConfiguration จากนั้นก็ประกาศ NSURLSession โดยกำหนดค่าเป็น defaultConfiguration พร้อมกับกำหนด delegateเมื่อเสร็จจากขั้นตอนนี้ก็เหลือแค่กำหนด task ที่ต้องการใช้งาน จากโค้ดของโปรแกรมในบรรทัดที่ 20 ก็ได้ประกาศให้ dataTask ให้เป็น task ที่จะใช้งาน และไม่ว่าจะประกาศ task แบบใดก็ตาม หลังจากประกาศเสร็จแล้ว task จะมีสถานะเป็น suspend ซึ่งหมายถึงว่ายังไม่เริ่มทำงาน ดังนั้นจึงต้องสั่งให้เริ่มทำงานด้วยการเรียกเมธอด resume ในส่วนของ while loop ทำหน้าที่แค่รอ dataTask ทำงานจนเสร็จ และถ้าหากโปรแกรมรอการรับจากเซิฟเวอร์นานเกินไปก็ให้ยกเลิกการทำงาน เมื่อให้โปรแกรมทำงานจนเสร็จสิ้นก็จะได้ผลลัพธ์ดังนี้

Program 18.1 Output

```
{
    "firstName": "John",
    "lastName": "Smith",
    "age": 25
}
```

Download

โปรแกรมที่ได้เขียนไปเป็นการใช้ task แบบ data เราจะเขียนโปรแกรมอีกสักโปรแกรมเพื่อดาวน์โหลดไฟล์มาเก็บไว้ที่เครื่อง สิ่งที่ต้องเขียนเพิ่มเติมเมธอดของ NSURLSessionDownloadDelegate ให้กับคลาส MyDelegate นั่นเอง

Program 18.2

MyDelegate.m

```
- (void)URLSession:(NSURLSession *)session
      downloadTask:(NSURLSessionDownloadTask *)downloadTask
didFinishDownloadingToURL:(NSURL *)location
{
    NSFileManager *fileMgr = [NSFileManager defaultManager];
    NSArray *URLs = [fileMgr URLsForDirectory:NSDesktopDirectory
                                    inDomains:NSUserDomainMask];

    NSURL *documentsDirectory = [URLs objectAtIndex:0];
    NSString *originalURL = [[[downloadTask originalRequest] URL] lastPathComponent];
    NSURL *destinationURL = [documentsDirectory URLByAppendingPathComponent:originalURL];
    NSError *errorCopy;

    BOOL success = [fileMgr copyItemAtURL:location toURL:destinationURL error:&errorCopy];
    if (success)
        NSLog(@"Download done");
    else
        NSLog(@"Error during the copy: %@", [errorCopy localizedDescription]);
```

```
}

- (void)URLSession:(NSURLSession *)session
      downloadTask:(NSURLSessionDownloadTask *)downloadTask
      didWriteData:(int64_t)bytesWritten
 totalBytesWritten:(int64_t)totalBytesWritten
totalBytesExpectedToWrite:(int64_t)totalBytesExpectedToWrite
{

}

- (void)URLSession:(NSURLSession *)session
      downloadTask:(NSURLSessionDownloadTask *)downloadTask
 didResumeAtOffset:(int64_t)fileOffset
expectedTotalBytes:(int64_t)expectedTotalBytes
{

}
```

โค้ดที่ได้เพิ่มเข้าไปมีด้วยกัน 3 เมธอด โดยเมธอดแรกนั้น จะถูกเรียกเมื่อการทำงานของ download task ได้เสร็จสิ้น ในเมธอดนี้ได้เขียนโค้ดเพื่อคัดลอกไฟล์มาไว้ยัง Desktop ของผู้ใช้งาน เนื่องจากเมื่อ download task ได้เริ่มต้นทำงานก็จะสร้าง temp file เพื่อใช้สำหรับเขียนข้อมูล ดังนั้นจึงเขียนโค้ดให้คัดลอกข้อมูลจาก temp file มาเก็บไว้นั่นเอง ส่วนอีกเมธอดเป็นเมธอดที่บอกจำนวนของข้อมูลที่ได้เขียนเสร็จ และเมธอดที่อยู่ล่างสุดเป็นส่วนที่เกี่ยวกับการจัดการ task ในกรณีที่ต้องทำงานต่อจากงานที่ค้างไว้ (resume) เมื่อเพิ่มโค้ดของ download delegate เสร็จเรียบร้อยแล้ว ในส่วนของโปรแกรมหลักให้เปลี่ยน NSURLSessionDataTask เป็น NSURLSesssionDownloadTask ดังนี้

main.m

```
        NSURLSessionDownloadTask* task = [session downloadTaskWithURL:url];
        [task resume];
```

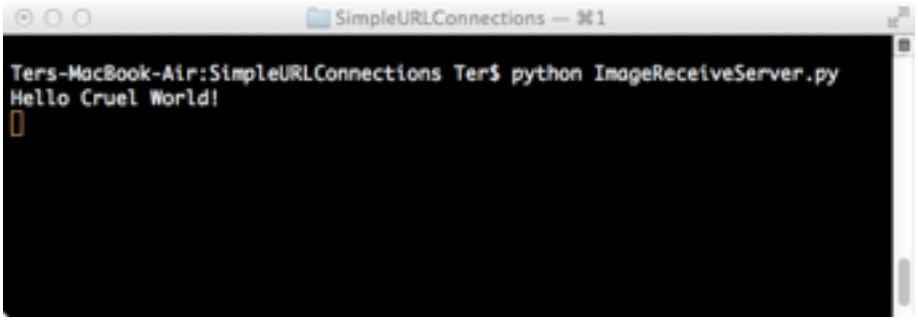
เมื่อโปรแกรมทำงานก็จะดาวน์โหล smith.json ไฟล์มาไว้ที่ desktop พร้อมกับแสดงผลที่ console ดังนี้

Program 18.2 Output

```
Download done
```

Upload

สองโปรแกรมที่ผ่านมาเป็นโปรแกรมที่รับข้อมูลจากเซิฟเวอร์ แต่โปรแกรมที่จะได้เขียนต่อไปนี้ เป็นโปรแกรมเพื่อใช้สำหรับส่งรูปภาพไปยังเซิฟเวอร์ เนื่องจากต้องมีเซิฟเวอร์สำหรับการ upload ดังนั้นก่อนที่จะเริ่มลงมือเขียนโค้ด เราจะใช้เครื่อง Mac ที่มีอยู่แล้วทำหน้าที่เป็น server สำหรับโค้ดของโปรแกรมเซิฟเวอร์ที่จะใช้ในโปรแกรมนี้ สามารถโหลดได้จาก https://developer.apple.com/library/ios/samplecode/SimpleURLConnections/SimpleURLConnections.zip หลังจากโหลดไฟล์เสร็จก็ unzip ให้เรียบร้อย จากนั้นจะพบกับไฟล์ ImageReceiveServer.py ซึ่งเป็นโปรแกรมเซิฟเวอร์อย่างง่ายที่เขียนด้วยภาษา pythone จากนั้นให้เปิดโปรแกรม terminal และสั่งให้เซิฟเวอร์ทำงานได้ด้วยคำสั่ง python ImageReceiveServer.py ดังที่แสดงดังรูป (กด ctr+c เพื่อสั่งให้เซิฟเวอร์หยุดทำงาน)



เมื่อสั่งให้เซิฟเวอร์ทำงานได้แล้ว ในลำดับต่อไปก็จะเขียนโปรแกรมสำหรับการ upload file ซึ่งมีโค้ดโปรแกรมดังนี้

Program 18.3

MyDelegate.m

```
- (void)URLSession:(NSURLSession *)session task:(NSURLSessionTask *)task
   didSendBodyData:(int64_t)bytesSent
    totalBytesSent:(int64_t)totalBytesSent
totalBytesExpectedToSend:(int64_t)totalBytesExpectedToSend
{
    NSLog(@"send: %lli , total send: %lli , total byte: %lli"
          , bytesSent , totalBytesSent ,totalBytesExpectedToSend);
}
```

ในส่วนของ delegate นี้มีเมธอดที่ต้องเขียนเพียงหนึ่งเมธอดเท่านั้น ซึ่งเป็นเมธอดที่บอกจำนวนข้อมูลที่ได้ส่งไปยังเซิฟเวอร์ในแต่ละครั้ง และโค้ดในส่วนของโปรแกรมหลักมีดังนี้

main.m

```
#import <Foundation/Foundation.h>
#import "MyDelegate.h"

int main(int argc, const char * argv[])
{

    @autoreleasepool {

        NSURL* postURL = [NSURL URLWithString:@"http://localhost:9000/cgi-bin/PostIt.py"];
        NSURL* fileURL = [NSURL fileURLWithPath:@"untitled.png"];
        MyDelegate* dataDelegate = [[MyDelegate alloc] init];

        // Build the request body
        NSString *boundary = @"boundary-demo";

        NSURLSessionConfiguration* config = [NSURLSessionConfiguration
                                             defaultSessionConfiguration];
        config.HTTPAdditionalHeaders = @{@"Content-Type":
                [NSString stringWithFormat:@"multipart/form-data; boundary=%@", boundary]};

```

เริ่มต้นด้วยการประกาศ url เซิฟเวอร์ที่จะส่งข้อมูลไป ในส่วนของ session configuration ได้กำหนดค่า http header ซึ่งเป็นข้อมูลที่จะถูกส่งไปยังเซิฟเวอร์ ได้ด้วยการกำหนดพร๊อพเพอร์ตี้ HTTPAdditionalHeader นอกจากการกำหนดค่า http header ใน session configuration ยังสามารถกำหนดใน NSURLRequest ได้อีกทาง แต่การกำหนดด้วย session configuration จะเป็นการกำหนดค่าเริ่มต้น ให้กับทุก request ที่ทำงานใน session นี้ ส่วนในกรณีที่กำหนดให้กับ NSURLRequest จะเป็นการกำหนดเฉพาะเจาะจง request ที่ต้องการ

```
21          // Body part for the attachament.
22          NSMutableData *body = [NSMutableData data];
23          NSData* imageData = [NSData dataWithContentsOfURL:fileURL];
24          if (imageData)
25          {
26              [body appendData:[[NSString stringWithFormat:@"--%@\r\n", boundary]
27                                dataUsingEncoding:NSUTF8StringEncoding]];
28
29              [body appendData:[[NSString stringWithFormat:@"Content-Disposition: form-data; "
30                                 "name=\"fileContents\"; filename=\"%@\"\r\n",
31                                 [fileURL lastPathComponent]]
32                                dataUsingEncoding:NSUTF8StringEncoding]];
33
34              [body appendData:[@"Content-Type: image/png\r\n\r\n"
35                                dataUsingEncoding:NSUTF8StringEncoding]];
36
37              //This is an image.
38              [body appendData:imageData];
39
40              [body appendData:[[NSString stringWithFormat:@"\r\n"]
41                                dataUsingEncoding:NSUTF8StringEncoding]];
42          }
43          [body appendData:[[NSString stringWithFormat:@"--%@--\r\n", boundary]
44                            dataUsingEncoding:NSUTF8StringEncoding]];
```

โค้ดในลำดับต่อมาเริ่มด้วยการเตรียมข้อมูลที่จะส่งไปยังเซิฟเวอร์ ซึ่งประกอบไปด้วยข้อมูลของโพรโทคอล http/1.1 เช่น content-type รวมไปถึงข้อมูลรูปภาพที่จะส่งไป จากนั้นก็เริ่มเขียนโค้ดที่ใช้สำหรับส่งข้อมูล

```
47          NSMutableURLRequest *request = [NSMutableURLRequest requestWithURL:postURL];
48          request.HTTPMethod = @"POST";
49
50          NSURLSession *session = [NSURLSession sessionWithConfiguration:config
51                                                                delegate:dataDelegate
52                                                           delegateQueue:nil];
53
54          NSURLSessionUploadTask* task = [session uploadTaskWithRequest:request fromData:body];
55          [task resume];
56
57
58          NSDate *start = [NSDate date];
59          while (task.state == NSURLSessionTaskStateRunning)
60          {
61              NSTimeInterval timeInterval = [start timeIntervalSinceNow];
62              if( fabs(timeInterval) > 15)
63              {
64                  NSLog(@"Server response time is too long");
65                  [task cancel];
66              }
67          }
```

เมื่อดูจากโค้ดตั้งแต่บรรทัดที่ 47 จะเห็นว่ามีรูปแบบเหมือนกับการเขียน download แต่สิ่งที่เพิ่มเข้ามาคือ NSURLRequest ที่ระบุว่าใช้ POST สำหรับส่งข้อมูล ส่วนการสร้าง task ที่จะทำงานใช้เมธอด uploadTaskWithRequest: fromeData พร้อมกับส่งค่าพารามิเตอร์ด้วยข้อมูลที่ได้เตรียมไว้ ก็เป็นอันเสร็จสิ้น เมื่อให้โปรแกรมทำงาน โปรแกรมทางฝั่งเซิฟเวอร์ก็จะแสดงข้อมูลดังรูป

และข้อมูลรูปภาพจะถูกเก็บไว้ยังโฟลเดอร์ images ซึ่งเป็น sub folder ภายในโฟลเดอร์ที่เซิฟเวอร์ได้ทำงาน

Program 18.3 Output

```
send: 24148 , total send: 24148 , total byte: 24148
```

NSURLSession with Block

ทั้งสามโปรแกรมที่ได้เขียนไปใช้ delegate ในการทำงานทั้งหมด ถ้าหากไม่ได้กำหนด delegate ให้กับ session ก็สามารถใช้ block แทนได้เช่นเดียวกัน เราจะเขียนโปรแกรมเพื่อใช้ download file ซึ่งมีการทำงานเหมือนกับโปรแกรม 18.2

Program 18.4

main.m

```
#import <Foundation/Foundation.h>

int main(int argc, const char * argv[])
{

    @autoreleasepool {

        NSString* json = @"https://dl.dropboxusercontent.com/u/7585756/smith.json";
        NSURL* url = [NSURL URLWithString:json];

        NSURLSessionConfiguration* config = [NSURLSessionConfiguration
                                             defaultSessionConfiguration];

        NSURLSession *session = [NSURLSession sessionWithConfiguration:config
                                                              delegate:nil
                                                         delegateQueue:nil];

        NSURLSessionDownloadTask* task = [session downloadTaskWithURL:url
           completionHandler:^(NSURL *location, NSURLResponse *response, NSError *error)
        {

                NSFileManager *fileMgr = [NSFileManager defaultManager];
                NSArray *URLs = [fileMgr URLsForDirectory:NSDesktopDirectory
                                                inDomains:NSUserDomainMask];

                NSURL *documentsDirectory = [URLs objectAtIndex:0];
                NSString *file = [json lastPathComponent];
                NSURL *dest = [documentsDirectory URLByAppendingPathComponent:file];
                NSError *errCopy;

                BOOL success = [fileMgr copyItemAtURL:location toURL:dest error:&errCopy];
                if (success)
                    NSLog(@"Download done");
                else
                    NSLog(@"Error during the copy: %@", [errCopy localizedDescription]);
```

```
36
37          }];
38          [task resume];
39
40          NSDate *start = [NSDate date];
41
42          // Wait until the task is finished
43          while (task.state == NSURLSessionTaskStateRunning)
44          {
45              NSTimeInterval timeInterval = [start timeIntervalSinceNow];
46              if( fabs(timeInterval) > 15)
47              {
48                  NSLog(@"Server response time is too long");
49                  [task cancel];
50              }
51          }
52
53
54      }
55      return 0;
56  }
```

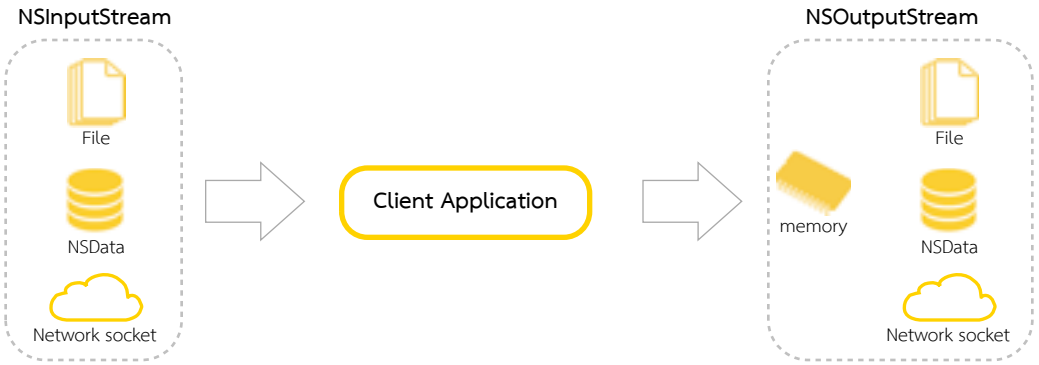
โปรแกรม 18.4 กำหนด delegate เป็น nil หรือไม่มี delegate และการสร้าง task จะใช้เมธอด downloadTaskWithURL: completionHandler: พร้อมโค้ดในส่วนของ completion block ซึ่งจะถูกใช้งานหลังจากที่ task ได้ทำงานเสร็จสิ้นแล้ว หากดูจากโค้ดของ block จะเห็นว่าเป็นการคัดลอกไฟล์มาไว้ยัง desktop ของผู้ใช้งาน ซึ่งมีการทำงานเช่นเดียวกับ delegate ที่ได้เขียนไปในโปรแกรม 18.2 นั่นเอง

Program 18.4 Output

```
Download done
```

NSStream

โปรแกรม iOS หรือ Mac OS X โดยทั่วไปแล้วจะใช้โพรโทคอล HTTP ในการรับและส่งข้อมูลกับทาง web sever เช่นโปรแกรมได้เขียนกันไปก่อนหน้านี้ แต่อย่างไรก็ตามในบางครั้งอาจจะมีความจำที่ต้องใช้งานในระดับต่ำกว่า HTTP แน่นอนว่าคลาส NSURLSession ไม่สามารถใช้กับงานลักษณะแบบนี้ได้ โชคดีที่ cocoa มีคลาส NSStream ที่ได้ออกแบบมาเพื่อใช้กับงานลักษณะดังกล่าว ทำให้ไม่ต้องเขียนการทำงานในระดับต่ำๆเช่น C Socket API และอย่างเช่นเคย ก่อนที่จะเขียนโปรแกรม ควรทำความเข้าใจกับโครงสร้างการทำงานของ NSStream เสียก่อน ดังที่ได้แสดงไว้ในรูป



จากรูปจะเห็นว่าไม่มีคลาส NSStream เพราะเนื่องจากคลาสนี้เป็นเพียง abstract class ส่วนคลาสที่ใช้งานจริงคือ NSInputStream ที่ทำหน้าที่ที่ใช้รับข้อมูล จากภาพจะเห็นว่า NSInputStream นอกจากจะรับข้อมูลผ่านทาง network แล้วยังสามารถรับข้อมูลจากไฟล์ หรือ NSData ได้อีกด้วย ส่วนคลาสที่ทำหน้าที่ส่งข้อมูลก็คือ NSOutputSteam ซึ่งสามารถส่งข้อมูลผ่านทาง network หรือส่งไปยังไฟล์ , หน่วยความจำ และ NSData ได้เช่นกัน
โปรแกรมที่จะได้เขียนต่อไป จะใช้คลาส NSStream เชื่อมต่อกับ echo server ที่มีการทำงานง่ายๆคือ เมื่อ client ส่งข้อความไปยังเซิฟเวอร์ ทางฝั่งเซิฟเวอร์ก็จะส่งข้อความนั้นกลับมาหา client นั่นเอง เราจะดาวน์โหลด echo server ซึ่งเขียนด้วยภาษา python ดังเช่นโปรแกรมที่ผ่านมา ได้จาก https://github.com/macfeteria/Objective-C-Demo/raw/master/Chapter%2018/Program%2018.5/echoServer.py เมื่อดาวโหลดเสร็จเรียบร้อยแล้วก็สั่งให้เซิฟเวอร์ทำงาน ด้วยการใช้คำสั่ง python echoServer.py ผ่านทาง Terminal เช่นเดิม ในส่วนของโปรแกรมฝั่ง client มีโค้ดดังนี้

Program 18.5

EchoClient.h

```
#import <Foundation/Foundation.h>

@interface EchoClient : NSObject <NSStreamDelegate>
{
    NSOutputStream*  outputStream;
    NSInputStream*  inputStream;
}
@property (assign) BOOL isDone;
-(void) initCommunitcation;
-(void) sendMessage:(NSString*) message;

@end
```

ในส่วนของ interface ได้ประกาศ output stream และ input stream ไว้ใช้งาน พร้อมกับเมธอดสำหรับเชื่อมต่อเซิฟเวอร์ และเมธอดที่จะใช้สำหรับส่งข้อมูล

EchoClient.m

```
#import "EchoClient.h"

@implementation EchoClient
-(void) initCommunitcation
{
    NSHost *host = [NSHost hostWithAddress:@"localhost"];

    NSInputStream __autoreleasing *iStream = nil;
    NSOutputStream __autoreleasing *oStream = nil;

    [NSStream getStreamsToHost:host
                          port:5000
                   inputStream:&iStream
                  outputStream:&oStream];

    outputStream = oStream;
    inputStream = iStream;

    [inputStream setDelegate:self];
    [outputStream setDelegate:self];

    [inputStream scheduleInRunLoop:[NSRunLoop currentRunLoop]
                           forMode:NSDefaultRunLoopMode];
    [outputStream scheduleInRunLoop:[NSRunLoop currentRunLoop]
                            forMode:NSDefaultRunLoopMode];

    [inputStream open];
```

```
    [outputStream open];

}
```

เมธอด initCommunication นี้ทำหน้าหลักคือ สร้างอ็อบเจ็ก inputStream และ outputStream และในการสร้างอ็อบเจ็กสองอย่างนี้จะไม่ได้ใช้การ alloc เหมือนอ็อบเจ็กทั่วไป แต่จะใช้คลาสเมธอด NSStream ในการสร้าง เนื่องจากเมธอดนี้ต้องการพารามิเตอร์ที่เป็นแบบ autorelease แต่ตัวแปรที่เราได้ประกาศ นั้นเป็น strong (อ็อบเจ็กที่สร้างด้วยคลาสเมธอด หรือมีการลักษณะแบบนี้ เช่น NSError จะเป็น autorelease ทั้งหมด) ดังนั้นแล้ว จึงต้องประกาศตัวแปร oStream และ iStream ที่เป็น autorelease ขึ้นมาใช้งานเพิ่มเติม จากนั้นค่อยส่งค่าให้กับ inputStream และ outputStream ในภายหลัง เมื่อสร้างอ็อบเจ็กเสร็จเรียบร้อย ก็กำหนด run loop ที่จะใช้งาน ก็เป็นอันเสร็จสิ้น

```
-(void) sendMessage:(NSString*) message
{
    self.isDone = NO;
    NSData *data = [message dataUsingEncoding:NSUTF8StringEncoding];
    [outputStream write:[data bytes] maxLength:[data length]];
}
```

เมธอด sendMessage นี้ทำหน้าที่ในการส่งข้อมูลผ่านทาง outputStream ส่วนการรับข้อมูลจะเขียนเมธอด stream:handleEvent ซึ่งเป็น delegate method ของ inputStream

```
- (void)stream:(NSStream *)theStream handleEvent:(NSStreamEvent)streamEvent
{
    NSLog(@"stream event %lu", streamEvent);
    switch (streamEvent)
    {
        case NSStreamEventOpenCompleted:
            NSLog(@"Stream opened");
            break;
        case NSStreamEventHasBytesAvailable:
        if (theStream == inputStream)
        {
            uint8_t buffer[1024];
            NSInteger len;
            while ([inputStream hasBytesAvailable])
            {
                len = [inputStream read:buffer maxLength:sizeof(buffer)];
                if (len > 0)
                {
                    NSString *output = [[NSString alloc]
                                        initWithBytes:buffer
                                        length:len
                                        encoding:NSASCIIStringEncoding];

                    if (nil != output)
                        NSLog(@"server said: %@", output);
                }
            }
            self.isDone = YES;
        }
            break;
        case NSStreamEventErrorOccurred:
            NSLog(@"Can not connect to the host!");
            self.isDone = YES;
            break;
        case NSStreamEventEndEncountered:
            [theStream close];
            [theStream removeFromRunLoop:[NSRunLoop currentRunLoop]
                                 forMode:NSDefaultRunLoopMode];
            theStream = nil;
            self.isDone = YES;
            break;
```

```
80              default:
81                  NSLog(@"Unknow Event:%lu",streamEvent);
82          }
83      }
84      @end
```

เมื่อมีเหตุการณ์ต่างๆเกิดขึ้น เช่น ติดต่อเซิฟเวอร์ผิดพลาด หรือมีข้อมูลเข้ามายัง inputStream เมธอดนี้ก็จะถูกเรียกใช้งาน ซึ่งจะได้รับ streamEvent เพื่อบอกเหตุการณ์หรือสิ่งที่เกิดขึ้นของ stream ในเมธอดนี้สิ่งที่เราเราสนใจจริงๆ ก็คือ NSStreamEventHasBytesAvailable นั่นคือมีข้อมูลเข้ามา เมื่อได้รับข้อมูล ก็ให้ inputStream ทำการอ่านข้อมูล ส่วนกรณีอื่นๆเช่น NSStreamEventEndEncountered นี้จะเกิดขึ้นในกรณีที่การติดต่อกับทางเซิฟเวอร์ได้สิ้นสุดลง ซึ่งเราได้เขียนโค้ดให้หยุดการทำงานของ stream นั่นเอง

Summary

โปรแกรมที่ได้เขียนไปในบทนี้เป็นเพียงแค่ตัวอย่างการใช้งานพื้นฐานเท่านั้น ไม่อาจจะครอบคลุมการใช้งานคลาสต่างๆที่เกี่ยวข้องกับ Network ได้ทั้งหมด อย่างไรก็ตามผู้เขียนเชื่อว่า ตัวอย่างโปรแกรมที่ได้เขียนไปเพียงพอสำหรับการทำความเข้าใจหลักการทำงานของคลาสต่างๆของระบบ Network ได้โดยรวม สำหรับผู้ที่สนใจศึกษาเพิ่มเติมเช่น การใช้ cache , cookie , ssl หรือ authentication สามารถศึกษาได้โดยตรงจากการอ่านเอกสารของแอปเปิ้ล ในบทหน้าเราจะเรียนรู้เทคโนโลยีสำคัญของ Mac OS X และ iOS นั่นคือ Grand Central Dispatch