MapKit revisited

History:

- Apple introduced MapKit with iOS, maps were based on Google.
- With iOS 6.0, Apple provided its own mapping service, which lacked some quality, especially level-of-detail.
- With iOS 7 Apple opened up its MKMapView to potential other map provider.
MapKit revisited

Techniques:

- Maps are loaded based on strict locative informations.
- Dedicated SDKs from third party content provider.
- Maps are loaded based on encoded informations, namely from map-tile-services.
A complete map can be loaded based on locative informations:

- One location with latitude and longitude and a bounding box
- Two locations forming a rectangular section.
- A textual address is used.

The map is loaded as described. Modifying such a map means always recalculating the rectangular section.

This style is used for static maps. It is not suitable for dynamic maps with panning and zooming.
SDKs

- Google Maps
  Typically there is a subclass, or a similar class to MKMapView, which should be used instead. Usually the delegate-pattern with the same methods as from MKMapView is used.

- Bing (Microsoft)

- MapQuest

- MapBox
  Third party SDK may show different concepts of the UI. Integration may be difficult.

- … and more
Requirements:

- Consistent projection scheme.
- Tiles are encoded by a tile scheme
  ✦ The scheme is used for loading tiles …
  ✦ … and displaying the tiles
MapKit revisited

Mercator projection

- True scale only on the equator
- Angles true on small scales
- Easy to use for rectangular tile schemes


Source: Google
MapKit revisited

[ Openstreetmap convention ]

- Tiles are 256 × 256 pixel PNG files
- Each zoom level is a directory, each column is a subdirectory, and each tile in that column is a file
- Filename(url) format is /zoom/x/y.png
- Zoom levels between 0 and 18, maybe more or less
- Zoom level n: $2^n \times 2^n$ tiles for the complete globe
MapKit revisited

Good News

- All major map-services are following the scheme:
  - Tiles 256 x 256 pixels
  - Mercator projection
  - Zoom levels
  - Same tile scheme; only Bing uses quad keys.

It’s really simple
MapKit revisited

staticNSString *consttemplate = @"http://tile.openstreetmap.org/{z}/{x}/{y}.png";

MKTileOverlay *overlay = [[MKTileOverlay alloc] initWithURLTemplate:template];
overlay.canReplaceMapContent = YES;

[self.mapView addOverlay:overlay
 level:MKOverlayLevelAboveLabels];

#pragma mark - MKMapViewDelegate

-(MKOverlayRenderer *)[mapView:(MKMapView *)mapView
 rendererForOverlay:(id <MKOverlay>)overlay
 {
   if ([overlay isKindOfClass:[MKTileOverlay class]]) {
     return [[MKTileOverlayRenderer alloc] initWithTileOverlay:overlay];
   }
   return nil;
 }
MapKit revisited

- Add informations above the tiles
  - subclass MKTileOverlay

- Watermarking
  - subclass only MKTileOverlayRenderer
MapKit revised

- (void)loadTileAtPath:(MKTileOverlayPath)path result:(void (^)(NSData *, NSError *))result {

    CGSize sz = self.tileSize;
    CGRect rect = CGRectMake(0, 0, sz.width, sz.height);

    UIGraphicsBeginImageContext(sz);
    CGContextRef ctx = UIGraphicsGetCurrentContext();
    [[UIColor grayColor] setStroke];
    CGContextSetLineWidth(ctx, 0.5);
    CGContextStrokeRect(ctx, CGRectMake(0, 0, sz.width, sz.height));
    NSString *text = [[NSString stringWithFormat:@"X=%ld\nY=%ld\nZ=%ld", (long)path.x, (long)path.y, (long)path.z] drawInRect:rect withAttributes:@{NSFontAttributeName:[UIFont systemFontOfSize:20.0], NSForegroundColorAttributeName:[UIColor blackColor]}];
    [text drawInRect:rect withAttributes:@{NSFontAttributeName:[UIFont systemFontOfSize:20.0], NSForegroundColorAttributeName:[UIColor blackColor]}];

    UIImage *tileImage = UIGraphicsGetImageFromCurrentImageContext();
    UIGraphicsEndImageContext();
    NSData *tileData = UIImagePNGRepresentation(tileImage);
    result(tileData, nil);
}

subclass of MKTileOverlay
MapKit revisited

Watermarking

-(void)drawMapRect:(MKMapRect)mapRect zoomScale:(MKZoomScale)zoomScale inContext:(CGContextRef)context {
  [super drawMapRect:mapRect zoomScale:zoomScale inContext:context];
  CGRect rect = [self rectForMapRect:mapRect];
  CGContextSetFillColorWithColor(context, [UIColor colorWithRed:1.0 green:0.5 blue:0.5 alpha:0.2].CGColor);
  CGContextFillRect(context, rect);
}

subclass MKTileOverlayRenderer implement drawMapRect:...
MapKit revisited

Offline

- The tiles from map-services are stored in instances of NSDate
- Instances of MKTileOverlay provide these instances.

What can we do here?
- Cache them locally using NSCache.
- Store them persistently.

There is nothing to do with Apple’s service.
- (void)loadTileAtPath:(MKTileOverlayPath)path result:(void (^)(NSData *data, NSError *error))result
{
    if (!result) {
        return;
    }

    NSString *keyPath = [self stringFromTileOverlayPath:path];
    NSPurgeableData *cachedData = [self.cache objectForKey: keyPath];
    if (cachedData) {
        result([NSData dataWithData: cachedData], nil);
    } else {
        NSURLRequest *request = [NSURLRequest requestWithURL: [self URLForTilePath: path]
            cachePolicy: NSURLRequestReloadIgnoringCacheData
            timeoutInterval: 20];
        [NSURLConnection sendAsynchronousRequest: request queue: self.operationQueue
            completionHandler: ^(NSURLResponse *response, NSData *data, NSError *connectionError)
            {
                NSPurgeableData *cachedData = nil;
                if (data) {
                    cachedData = [NSPurgeableData dataWithData: data];
                    [self.cache setObject: cachedData forKey: keyPath];
                    [self saveTile: data toFileSystemWithTilePath: keyPath];
                }
                result(data, connectionError);
            }];
    }
}
MapKit revisited

Persistent storage

Collect tiles while connected and use them offline.

Take care of the MKTileOverlayPath

Store the tiles using CoreData
Using the file-system

Read the license(s)!
MapKit revisited

Some demo, maybe
MapKit revisited

Customization

✧ Tiles from different sources can be combined according to

✧ scale, or zoom-level

✧ location

✧ user dependent data

✧ Other overlays can be added:

✧ as map-tiles

✧ as shapes
MKMapSnapshotOptions *options = [[MKMapSnapshotOptions alloc] init];
options.region = self.mapView.region;
options.size = self.mapView.frame.size;
options.scale = [[[UIScreen mainScreen] scale];

NSURL *fileURL = [[NSURL fileURLWithPath:@"path/to/snapshot.png"]);

MKMapSnapshotter *snapshotter = [[MKMapSnapshotter alloc] initWithOptions:options];
[startWithCompletionHandler:^ (MKMapSnapshot *snapshot, NSError *error) {
    if (error) {
        NSLog(@"[Error] %@", error);
        return;
    }

    UIImage *image = snapshot.image;
    NSData *data = UIImagePNGRepresentation(image);
    [data writeToURL:fileURL atomically:YES];
}];

Does not draw annotations
MapKit revisited

- Using directions requires always a connection and/or some sort of registration.

Mapkit provides MKDirections and MKDirectionsRequest, provider is Apple.

Third party: MTDirectionsKit (usable before iOS 7.0)

Different provider, API-keys and/or registration is needed.

Once retrieved, directions can be shown on all kind of maps as overlays.
MapKit revisited

Ecosystem

A real ecosystem has been established in recent years, MapKit is one part of it.

Some options:

- Dedicated own Tile-server, e.g. ArcGis-Server
- Using MapBox, TileMill an so on
- Using vector-based maps with custom color schemes for renderings
MapKit revisited

Thank you!