

Workshop zum infoware-Navigation-API

GPS-Navigation selbst gebaut

Wer eine Navigationsanwendung für das iPhone bauen will ist auf Tools angewiesen. Am Beispiel des maptrip iPhone SDK zeigt mac-developer, wie es geht. **Thomas Schulte-Hillen**

Auf einen Blick

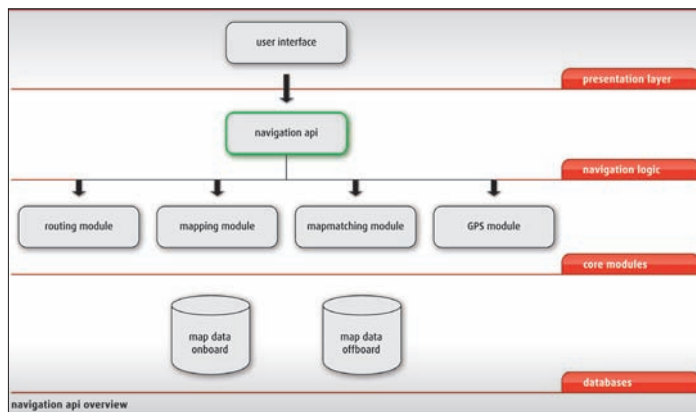
Inhalt

Das Navigations-SDK maptrip iPhone der Firma infoware bietet die gesamte Funktionalität für die Entwicklung einer GPS-Navigationsanwendung. Über ein übersichtliches und leicht zu benutzendes API können alle Navi-Funktionen von der Karte bis zur Zielführung zum Bau einer eigenen Anwendung oder zur Erweiterung bestehender Apps genutzt werden.

Auch Killer-Apps fangen mal klein an. Als 2002 die ersten Navigationsanwendungen als reine Softwarelösung zum Installieren auf PocketPCs auf den Markt kamen, war noch eine PC-Anwendung zum Ausschneiden eines Kartenkorridors erforderlich. Speicherplatz war knapp, also wurde nur dieser Korridor vom PC auf den mobilen Begleiter übertragen.

Inzwischen sind die Kinderkrankheiten überstanden und die Branche boomt. Aus vielen Navigationsfirmen wurden Hardwarehersteller. Erbitterte Preiskämpfe forderten so manches Opfer und lichtet die Reihen der Anbieter. Heute stürzen sich TomTom, Navigon und Co. wieder mit reinen Softwarelösungen auf das iPhone und den App Store. Googles und Nokias Vorstöße in die Navi-Welt vermitteln eine Ahnung davon, welche stürmische Entwicklung dieser Technologie auf Smartphones noch bevorsteht. Ein breites Spektrum von Anwendungen vom Kneipenführer bis zur Logistik wartet darauf, mit den hilfreichen Lotsenfunktionen ergänzt zu werden.

Aus technologischer Sicht ist die Turn-by-Turn-Navigation die Königsklasse der Location Based Services. Wer eine leistungsfähige Navi-



Die Architektur des SDK von der API-Schnittstelle bis zur Datenebene (Bild 1)

gation haben will, muss auf einem Gerät mit knappen CPU- und Speicherressourcen rund 173 Millionen Adressen und 56 Millionen Straßensegmente verwalten. Eine Route vom Nordkap nach Lissabon ist bitteschön in wenigen Sekunden zu berechnen, und wehe, die Navi macht auch nur eine falsche Ansage!

Navigation my way!

Als Software-Entwickler wünscht man sich oft mehr Flexibilität, als die Standard-Navis bieten. Denn eigentlich möchte man Navigationsfunktionen nach eigenen Vorstellungen in die Anwendung integrieren. Wer mit den monolithischen Standard-Navis und ihren mageren Schnittstellen nicht vorlieb nehmen möchte, braucht ein geeignetes Tool.

Das maptrip iPhone SDK

Das maptrip iPhone SDK implementiert alle für eine Turn-by-Turn-Navigationsanwendung nötigen Kernfunktionen und verbirgt diese hinter einem leicht verständlichen API. Für die Anwendungsprogrammierung ist kein Wissen um komplexe interne Vorgänge mehr nötig.

Aus technischer Sicht kommt das SDK als statische Library *iwNaviAPI.a* samt C-Header-Datei *maptripAPI.h* daher. Das Interface auf reiner C-Basis ist der Plattformunabhängigkeit des SDK geschuldet. Um die Anbindung für Objective-C-Programmierer zu erleichtern, gibt es darauf aufsetzend eine Objective-C-Wrapper-Schicht, die das C-Interface eins zu eins abbildet, jedoch unter Ausnutzung der Sprachfeatures von Objective-C, wodurch die Handhabung einfacher wird.

Die Objective-C-Schnittstelle untergliedert sich in drei Header-Dateien:

- **maptripApiWrapper.h** – API-Methoden.
- **maptripApiDelegates.h** – definiert das Protokoll des *maptripApiNavigationDelegate* und des *maptripApiGPSDelegate*.
- **maptripApiGeoAddress.h** – definiert eine Datenstruktur zum Austausch von Adressinformationen.

Neben dem eigentlichen Softwaremodul liegen dem SDK noch Kartendaten von Tele Atlas oder Navteq bei. Außerdem sind Sprachdateien für die Ansagegenerierung und eine Menge Konfigurationsdateien vorhanden, über die sich das Software Development Kit parametrisieren lässt.

Dieser Artikel stellt ein solches Tool vor, das maptrip iPhone SDK (www.infoware.de). Das Software Development Kit der Firma infoware kapselt alle kritischen Funktionen wie Routenberechnung oder Geocodierung, der Entwickler braucht sich darum nicht zu kümmern (Bild 1). Stattdessen stellt maptrip High-Level-Funktionen zur Verfügung, die der Entwickler braucht, um eine Navi-Anwendung im eigenen Stil zu erstellen. Die Ablaufsteuerung der Navigation, also die Geocodierung, Routenberechnung, Generierung von Ansagen und deren Synchronisierung mit der aktuellen GPS-Position sowie die Kartenanzeige, übernimmt maptrip. Die Benutzerschnittstelle ist davon losgelöst und kann vom Entwickler komplett individuell entworfen werden. Sowohl Layout und Grafik als auch die Benutzerführung kann nach eigenen Vorstellungen an den jeweiligen Anwendungsfall angepasst werden. Wie das geht, soll im Folgenden am Beispiel einer einfachen selbstgebauten Navi skizziert werden.

Bauplan einer Navi-Anwendung

Zur Demonstration der einzelnen SDK-Funktionen wird nun Schritt für Schritt eine Navigationsanwendung erstellt. Um den Rahmen dieses Artikels nicht zu sprengen, soll es bei einem ganz einfachen Workflow mit nur drei Screens bleiben:

- **Screen 1:** Zieleingabe mit Stadt, Straße und Hausnummer (Bild 2).
- **Screen 2:** Auswahl der Zieladresse (Bild 3).
- **Screen 3:** Kartendarstellung mit Fahrhinweisen (Bild 4).

Über den Button *Neues Ziel* kann von Screen 3 auf Screen 1 gewechselt und mit der Eingabe eines neuen Ziels begonnen werden.

Initialisieren des API

Die API-Klasse ist als Singleton durch die Methode *instance* realisiert. Das heißt, sie kann nur ein einziges Mal erzeugt werden. Bevor man die Funktionalität des Navigations-SDK nutzen kann, muss der *MaptripApiWrapper* initialisiert werden. Dies geschieht durch Aufruf der Methode *initWithPath*, welche als Parameter den Verzeichnispfad der Karteninstallation erwartet. Innerhalb dieses Aufrufs initialisiert die Library alle internen Komponenten, welche unter anderem Indexdateien des beiliegenden Kartenmaterials öffnen und intelligente Caches im Speicher anlegen. Vor dem Beenden der Applikation sollte die Wrapper-Klasse per *release* freigegeben werden, um keine Speicherlecks oder andere Relikte zu hinterlassen. Um die Einrichtung des *CLLocationManagers* muss man sich als Nutzer des API nicht kümmern, denn bereits während der Initialisierungsphase registriert sich das SDK für Positionsupdates beim System. Bei Bedarf erlaubt die Eigenschaft *GPSDelegate* das Empfangen der Positionsinformationen über das API.

Zieleingabe und Geocodierung

Bevor die erste Fahrt mit der Navigationsanwendung beginnen kann, muss der Zielort spezifiziert werden. Dafür wird aus den Adressbestandteilen Postleitzahl, Stadt, Straße und Hausnummer eine geografische Koordinate ermittelt, welche vom Navigationssystem als Zielpunkt verwendet wird (Geocodierung). Für die einzelnen Adressbestandteile kann es unterschiedliche Schreibweisen geben – etwa *Bonnerstr.*, *Bonnerstraße*, *Bonner Straße* oder *Bonner Strasse* –, weshalb die Geocodierung nicht ausschließ-

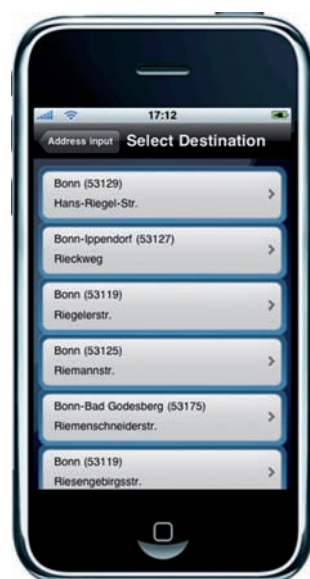
Autor



Thomas Schulte-Hillen ist Dipl.-Ing. Nachrichtentechnik und geschäftsführender Gesellschafter der Firma infoware GmbH. Infoware ist spezialisiert auf Entwickler-Tools für Navigation, Routenberechnung und Slippy-Map-Anwendungen. Im vergangenen Jahr hat das Unternehmen begonnen, sein Produktportfolio um Anwendungen für iPhone, Android und Blackberry zu erweitern.



Zieleingabe mit Stadt, Straße und Hausnummer (Bild 2)



Auswahl der Zieladresse (Bild 3)



Kartendarstellung mit Fahrhinweisen (Bild 4)

Listing 1: Die wichtigsten API-Methoden

```

@interface MaptripApiWrapper : NSObject
{ id<maptripApiNavigationDelegate> naviDelegate;
  id<maptripApiGPSDelegate> GPSDelegate;
}
@property (nonatomic, retain) id<maptripApiNavigationDelegate> naviDelegate;
@property (nonatomic, retain) id<maptripApiGPSDelegate> GPSDelegate;

+ (MaptripApiWrapper*)instance;
- (int)initWithPath:(NSString*)dataPath;
- (int)setMapViewFrame:(CGRect)frame:(int)orientation;
- (UIView*)getMapView;
- (int)setViewMode:(int)viewMode;
- (int)startNavigation;
- (int)stopNavigation;
- (int)setDestination:(GeoAddress*)destination;
- (int)startMapEngine:(bool)start;
- (int)geocodeAddress:(GeoAddress*)address;
- (GeoAddress*)getNextAddress;
@end

```

lich exakte Übereinstimmungen zulässt, sondern auch unscharfe Treffer als Ergebnis zurückliefert. Geringe Ungenauigkeiten bei der Eingabe werden somit automatisch korrigiert. Weiterhin ist zu beachten, dass das Ergebnis einer Geocodierung nicht eindeutig sein muss. Bei Eingabe des Stammwortes *Bonn* könnten bei der Straßensuche beispielsweise die *Bonner Straße* und der *Bonner Weg* in einer Stadt gefunden werden. Selbst bei der Angabe der vollständigen Adresse kann es denselben Straßennamen mehrfach geben, allerdings in unterschiedlichen Postleitzahlgebieten. Bereits anhand dieses Beispiels wird klar, dass man sich aufgrund der kombinatorischen Möglichkeiten bei der Eingabe auf mehr als ein Ergebnis einstellen muss. Deshalb wird Screen 2 für die Auswahl des Ziels aus der Trefferliste benötigt.

Die Geocodierung wird – wie aus [Listing 1](#) ersichtlich – durch die beiden Methoden *geocodeAddress* und *getNextAddress* realisiert.

Über die Methode *geocodeAddress* werden die Adressbestandteile in eine oder mehrere geocodierte Adressen aufgelöst. Der Rückgabewert

Listing 2: Adresseingabe

```

-(void)onButtonOk
{
    GeoAddress* address = [[GeoAddress alloc] init];
    address.city = [city copy];
    address.street = [street copy];
    address.house = [house copy];

    if (MTA_ERROR_OK == [maptripApiWrapper geocodeAddress: address])
    [self showResultList];
}

```

gibt Aufschluss darüber, ob der Vorgang erfolgreich war. Durch die Ergebnisliste kann im Anschluss daran über *getNextAddress* bequem iteriert werden, um Oberflächenelemente nach und nach mit Inhalt zu füllen. Für das Beispielprojekt ist ein einfacher Eingabedialog mit drei *text views* – für Stadt, Straße und Hausnummer – und einem *Los*-Button vorgesehen. Der Button ruft die API-Funktion zum Geocodieren auf ([Listing 2](#)).

Liefert die Routine einen gültigen Rückgabewert, wurden Adressen gefunden, welche dem Benutzer auf Screen 2 angezeigt werden. Dieser Screen besteht aus einer Table View, in der die Ergebnisse der Adresssuche angezeigt werden ([Listing 3](#)).

Zielauswahl und Start der Routenführung

Nachdem der Nutzer sich für einen Treffer aus der Zielliste entschieden hat, muss dieser zur Routenführung übergeben werden. Das geschieht über die API-Methode *setDestination*. Die Übergabe der Destination führt allerdings noch zu keiner Aktivität innerhalb der Engine. Erst der Aufruf von *startNavigation* bewirkt, dass eine aktive Routenführung beginnt. Alle Ereignisse, die während einer Routenführung auftreten, werden durch Auswertung der aktuellen GPS-Position getriggert. Verändert sich der Standort nicht, ist auch keine Aktualisierung der Routenführung notwendig. Im Takt der Positionsänderung signalisiert das API somit über Delegate-Methoden aktuelle Ereignisse, auf die im Kontext der Anwendung reagiert werden kann. Einen Überblick über die Arten von gemeldeten Ereignissen gibt [Listing 4](#).

Routenberechnung und Karte

Nachdem der Benutzer sein Ziel gewählt hat und die Routenführung gestartet wurde, wird in den Navigations-Screen (Screen 3) gewechselt. Dieser soll zum Großteil aus der Navigationskarte bestehen. Die Anzeige der Karte ist allerdings optional. Das SDK unterstützt auch die Pfeilnavigation sowie eine rein akustische Zielführung.

Für die Beispielanwendung sollen zusätzlich zur Karte die aktuellen Abbiegesymbole, die Distanz zum Ziel und die verbleibende Zeit bis zur Ankunft angezeigt werden ([Bild 4](#)).

Für die Realisierung von Screen 3 muss im Layout-Editor von Xcode ein großes *UIView*-Element für die Kartendarstellung vorgesehen werden. Die Kartenansicht inklusive Fahrzeugsymbol und aktueller Route wird per *getMapView* angefordert. Rückgabewert ist ein *UIView*-Objekt, welches als *Childview* in das im Layout vordefinierte Element integriert werden kann. An-

gezeigt wird je nach Konfiguration eine Karte in 3D- oder 2D-Darstellung. Im 2D-Modus wird noch zwischen *Fahrtrichtung* und *nordweisend* unterschieden (*setViewMode*). Die *UIView* des Kartenfensters realisiert bereits die Bedienbarkeit per Multitouch-Eingabe. So kann bequem mit Daumen und Zeigefinger die Zoomstufe des Viewers geändert werden. Die interne Komponente rendert die Karte durchgängig mit OpenGL ES (Bild 5 und Bild 6).

Ereignisgesteuerte Routenführung

Der Prozess der Routenführung ist ereignisbasiert. Wenn sich aufgrund einer Positionsänderung interne Zustände verändern, werden diese Ereignisse an die Anwendung weitergegeben.

Die Methode *onNewManeuver* signalisiert, wenn es Zeit für die Anzeige eines neuen Abbiegesymbols ist. Vor dem eigentlichen Abbiegekommando gibt es in der Regel zwei Hinweise, die den Fahrer auf das bevorstehende Manöver einstimmen. Je nach Straßenklasse und Fahrgeschwindigkeit variiert das Timing. Auf Autobahnen etwas früher, in der Stadt später. Der erste Parameter von *onNewManeuver* enthält den Typ des anzuzeigenden Symbols, der zweite die Entfernungsangabe für dieses Manöver, also etwa 0.3 für 300 Meter verbleibende Entfernung zum Manöverpunkt. Als Standardsymbole liegen dem SDK PNG-Dateien bei. Diese können aber durch eigene Grafiken ersetzt werden.

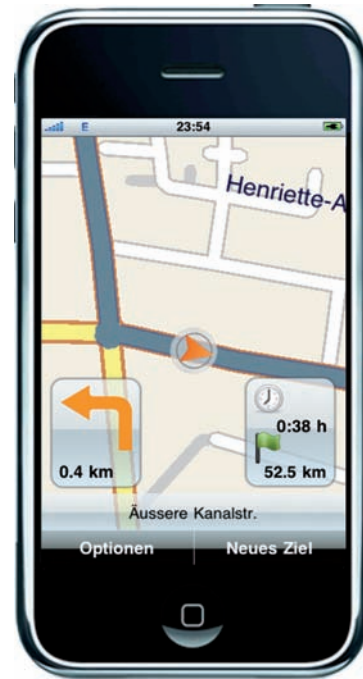
Die Verarbeitung einer neuen Position führt unmittelbar zur Änderung der Werte für die geschätzte Fahrtdauer und die verbleibende Distanz zum Ziel. Diese Werte werden über die Methode *onNewCrossingInfo* an die Anwendung geliefert und ständig aktualisiert. Dem Anwendungsentwickler bleibt lediglich die Aktualisierung der Anzeigeelemente in der Oberfläche. Will man dem Benutzer mehr Informationen geben, zeigt man zusätzlich den aktuellen Straßennamen an, der ebenfalls als Parameter mitgegeben wird. Zur zusätzlichen Orientierung für den Fahrer generiert das SDK per *onNewSignpost* zusätzliche Wegweiser für bestimmte Stellen – zum Beispiel vor Autobahnauf- und -abfahrten. Diese Information entspricht im Wesentlichen der realen Beschilderung im Straßenverkehr. Die Art der Beschilderung (Autobahn, Bundesstraße et cetera) wird über den Typ-Parameter signalisiert.

Ziel erreicht!

Wer den Anweisungen der Zielführung gefolgt ist, kommt schließlich an seinem Zielort an. In der Karte wird der genaue Zielpunkt durch einen hüpfend animierten Pfeil in 3D dargestellt – ein Vorgeschmack auf die kommenden Möglich-



Die gerenderte 2D-Darstellung der Karte (Bild 5)



2D-Darstellung, „nach Norden weisend“ (Bild 6)

keiten mit OpenGL. Die Zielerreichung wird der Anwendung vom API über die Delegate-Methode *onDestinationReached* mitgeteilt.

Fazit

Der Entwurf der kleinen Beispielanwendung zeigt, dass mit Hilfe des *maptrip* iPhone SDK mit relativ geringem Aufwand eine voll funktionsfähige Navigationsanwendung nach eigenen Bedürfnissen gebaut werden kann. [bl]

Listing 3: Auswahlliste füllen

```
-(void)showResultList
{
    GeoAddress* address;
    while (MTA_ERROR_OK == [maptripApiWrapper getNextAddress: address])
    {
        [self fillListItem:address];
    }
}
```

Listing 4: Die wichtigsten Delegates

```
@protocol maptripApiNavigationDelegate
-(void)onNavigationStarted;
-(void)onDestinationReached;
-(void)onNewRoute;
-(void)onRoutingProgress: (double)progress;
-(void)onNewManeuver: (ManeuverType)type >
    distance:(double)distanceToManeuver;
-(void)onNewSignpost: (NSString*)signPost type:(SignpostType)type;
-(void)onNewCrossingInfo: (NSString*)streetName >
    estimatedTimeOfArrival:(CFbsoluteTime)eta >
    distanceToDestination:(double)dtd;
@end
```