Oracle Endeca for Mobile
Getting Started Guide – iPhone Application

Copyright and Disclaimer

Getting Started Guide – iPhone Application

This guide assumes you have installed the endeca-ios-application.zip package on a Mac development environment (see the Install Guide). In order to release an iPhone application to the Apple App Store, you will also need to sign up and pay for the Apple Developer Program (http://developer.apple.com/iphone/).

Getting Started with iPhone Programming and Objective-C

If you are new to iPhone programming and Objective-C, it is highly recommended that you read the following guides on http://developer.apple.com/iphone.

1. Learning Objective-C: A Primer:
   http://developer.apple.com/iphone/library/referencelibrary/GettingStarted/Learning_Objective-C_A_Primer/
   Also, "The Objective-C Programming Language" document that is linked from this page is a great comprehensive guide to Objective C.

2. iOS Development Guide:
   http://developer.apple.com/iphone/library/documentation/Xcode/Conceptual/iphone_development/index.html

3. iOS Application Programming Guide:
   http://developer.apple.com/iphone/library/documentation/iPhone/Conceptual/iPhoneOSProgrammingGuide/index.html

4. Your First iOS Application:
   http://developer.apple.com/iphone/library/documentation/iPhone/Conceptual/iPhone101/index.html

5. View Controller Programming Guide:
   http://developer.apple.com/iphone/library/featuredarticles/ViewControllerPGforiPhoneOS/Introduction/Introduction.html

6. Table View Programming Guide:
   http://developer.apple.com/iphone/library/documentation/UserExperience/Conceptual/TableView_iPhone/AboutTableViewsiPhone/AboutTableViewsiPhone.html

Creating a New Project

1. Copy the "refapp" folder to a "myprojectname" folder at the same level

2. In the left menu in XCode, open Target, right-click "refapp", and click "Get Info". Select the Build tab at the top, then select "All Configurations" and "Settings Defined at This Level" from the 2 drop down boxes at the top.

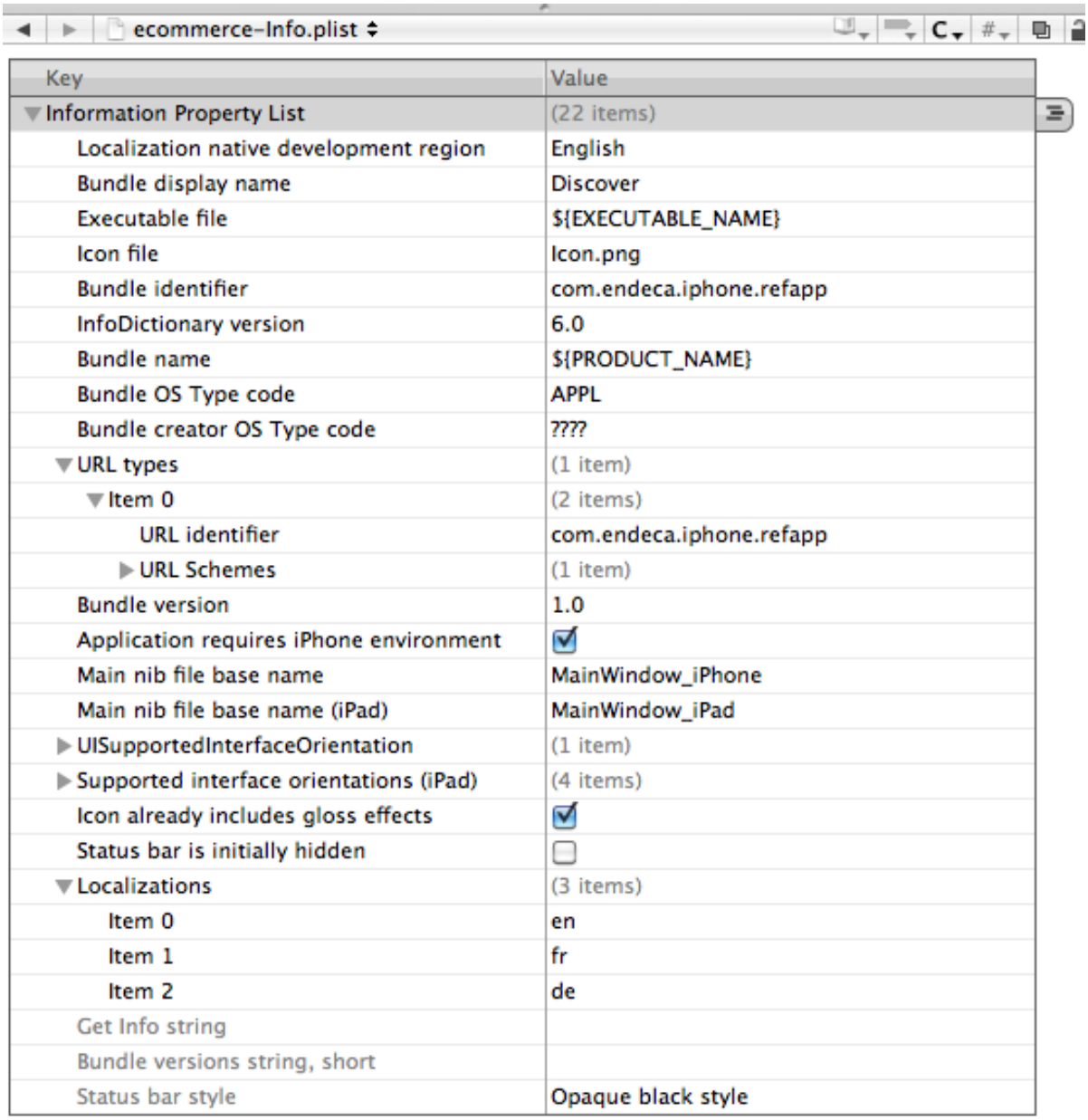
   Change Product Name to "myprojectname" of your choosing.

3. In the left menu in XCode, open the Resources click on ecommerce-Info.plist.

   1. Change the "Bundle identifier" – this will identify your app in the App Store, so should be unique from any other app in the store.
   2. Change the "Bundle display name" – this will be the text that shows up below the icon on

the home screen

3. Change the “URL identifier” within the URL types – this is used for launching the app from a url
4. If releasing an iPhone only app, it will still need to run on the iPad, so change “Main nib file base name (iPad)” to “MainWindow_iPhone”
5. If not localizing to any additional languages, remove the “fr” and “de” items from “Localizations”



4. Right-click "refapp" under Target and select Rename

   Change to myprojectname

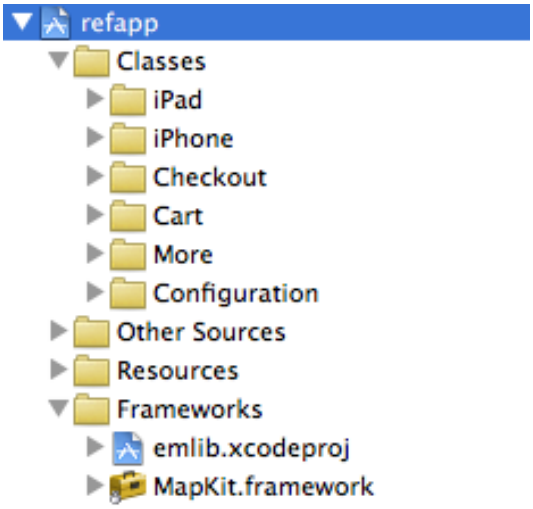
5. Close the app in XCode, and rename the Project file from refapp.xcodeproj to myprojectname.xcodeproj

Refapp Project Overview

The “refapp” project is setup as a Universal App, meaning you can submit one project to the Apple App Store, which will contain both an iPhone and iPad app, and based on which device the app is downloaded onto, will open the right version of the app accordingly.

The screenshot below shows the basic Group structure of the project, with iPad specific classes in the iPad folder, iPhone specific classes in the iPhone folder, and shared classes in Checkout, Cart, More and Configuration. The idea here is that certain views should be optimized for iPad or iPhone (e.g. main/home screen, browse views or landing pages, the product list and the detail page), but others views can use the same presentation on both devices (e.g. Cart, Checkout, Menus, Typeahead, Guided Navigation, Browse by Brand).

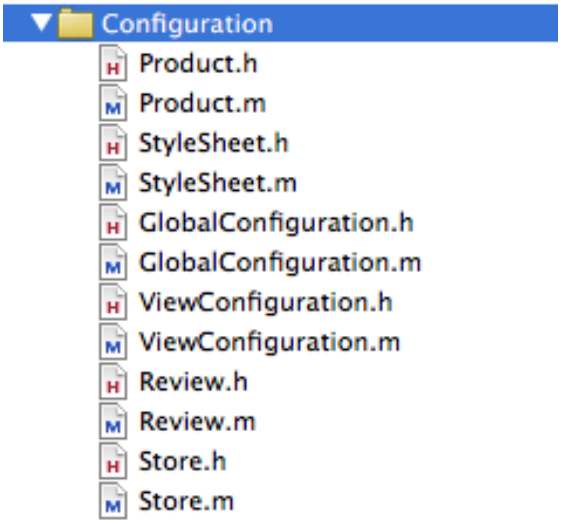
You can also see the reference to the emlib.xcodeproj, which is the core Oracle Endeca Mobile iOS library.



The Refapp project contains default implementations of a number of important configuration classes:

EMRecord, EMGlobalConfiguration, EMViewConfiguration, and EMDefaultStyleSheet.

These classes can be customized according to the specific needs of the application, and are extensible, so that the same configuration concepts can be leveraged by any custom components. The sections below walk through these classes in more detail.


Configuration
Product.h
Product.m
StyleSheet.h
StyleSheet.m
GlobalConfiguration.h
GlobalConfiguration.m
ViewConfiguration.h
ViewConfiguration.m
Review.h
Review.m
Store.h
Store.m

GlobalConfiguration.m

This file contains a number of important sections:

- Server Endpoints
- API Keys
- Model and DataSource config
- View Controller config
- Page Builder config

You will need to change at a minimum the endpoints, api keys, and sort config for a new application. The other sections allow you to override default settings and provide extension points if you want to add your own Page Builder powered views, add your own DataSource implementations against a different backend web service, or supply your own model classes.

1. Change the endpoint configurations to point to your Endeca Mobile API server. If all of your endpoints are shared you can simply change the following line at the top of GlobalConfiguration.m:

```
#define ENDPOINT @"http://[host]:[port]/mobile/"

- (NSString *)typeaheadEndpoint {
     return [NSString stringWithFormat:ENDPOINT
"search.getSuggestions.json?searchTerms=%@", @"%@"];
}

- (NSString *)detailEndpointForRecord:(EMRecord *)item {
     return [NSString stringWithFormat:ENDPOINT "detail.json?R=%@",
item.spec];
}

- (NSString *)searchEndpoint {
     return ENDPOINT "search/api.json";
}

- (NSString *)storesEndpoint {
     return ENDPOINT "stores/api.json";
}

- (NSString *)reviewsEndpoint {
     return ENDPOINT "reviews/api.json";
}

- (NSString *)cartEndpoint {
     return ENDPOINT "cart";
}

- (NSString *)brandsEndpoint {
     return ENDPOINT "brands.getList.json";
}
```

2. Update API Keys:

```
- (NSString *)bitlyLogin {
     return @"endeca";
}

- (NSString *)bitlyApiKey {
     return @"R_91993083f18b311e3f8a91edfba4ad56";
}

- (NSString *)facebookAppKey {
     return @"b436f06ce17bfd1e6d849dfe1a1ab15a";
```

```
}

- (NSString *)twitterOAuthConsumerKey {
      return @"UbQZZ8Mjn4q1vSDbwRq9Iw";
}

- (NSString *)twitterOAuthConsumerSecret {
      return @"b9g4vkDoUpShSFIz3K4ObIb2Kz8qJ4nGljVqIJvYfoM";
}
```

3. Update the searchQueryControllerFromDataSource: method.

```
controller.sortOptions = [NSArray arrayWithObjects:
  [EMSort sortWithKey:nil displayValue:@"Relevance"],
  [EMSort sortWithKey:@"product.price" order:SORT_ORDER_DESCENDING
displayValue:@"Price" momentary:YES],
  [EMSort sortWithKey:@"product.review.avg_rating" order:SORT_ORDER_DESCENDING
displayValue:@"Rating"],
  nil];
```

If you want to remove a predefined sort simply delete an object from the constructor ie. To remove the review sort you would delete: `[EMSort sortWithKey:@"product.review.avg_rating" order:SORT_ORDER_DESCENDING displayValue:@"Rating"],`

Product.m/Store.m/Review.m

Most of this class will need updating to use the right property keys for your dataset. This class is instantiated from a JSON dictionary that is returned from the search endpoint (in the case of search results), or the detail endpoint (in the case of a detail page). This dictionary is stored in the "_values" property, and the method implementations (e.g. "price", "title", "description") are responsible for pulling out the appropriate keys from this dictionary. These methods can also be used to provide some custom logic to manipulate the values before returning them. A couple examples are shown below:

```
- (NSString *) cartProductID {
      return [_values valueForKey:@"product.id"];
}

- (NSString *) listProductID {
      return [_values valueForKey:@"product.id"];
}
```

If the values are Dimensions, and not properties, you will need to pull them out differently, such as in the brandName method below. The value can also be an NSArray of either EMFacetValueSelection objects, or NSString objects if the Endeca property or dimension is multi-assign.

```
- (NSString *) brandName {
      id val = [_values valueForKey:@"product.brand.name"];
      if([val isKindOfClass:[EMFacetValueSelection class]]) {
            EMFacetValueSelection *brandDimVal = (EMFacetValueSelection
*)val;
            return brandDimVal.facetValue;
      } else {
            return (NSString *)val;
      }
}
```

The JSON dictionary returned by the Mobile API will return properties as either a single value, or

an array of value if there is more than one value. In order to make working with this potential for different data types on different records, the EMRecord class provides two convenience methods:

```
- (NSString *)firstValueForKey:(NSString *)key;

- (NSArray *)valueArrayForKey:(NSString *)key;
```

ViewConfiguration.m

This class provides a place to override the default TableViewCell implementations for View Controllers in the emlib component library. Most of the methods are not redefined in the Refapp, so you can look at EMViewConfiguration.m in emlib for the default implementations.

```
- (EMSearchTableCell *) listCellForRecord:(EMRecord *)record
reuseIdentifier:(NSString *)reuseIdentifier {
      EMProductListCell *cell = [[[EMProductListCell alloc]
initWithStyle:UITableViewCellStyleDefault
reuseIdentifier:reuseIdentifier] autorelease];
      cell.imageSize = 85;
      return cell;
}

- (NSInteger) listCellHeight {
      return 110;
}
```

StyleSheet.m

This makes use of the TTStyleSheet concept from the Three20 library, and allows you to override many of the styles that are used within the emlib Component library. It also provides an architecture for the separation of styles from the view and controller code of any custom components built on top of the Refapp.

For a full list of the properties that can be overridden, you should also look at EMDefaultStyleSheet in the emlib project.

To refer to a style defined in the StyleSheet.m or one a super class, use the “TTSTYLEVAR” macro:

```
    navController.navigationBar.tintColor =
TTSTYLEVAR(navigationBarTintColor);
```

Attribution Requirements for using the Twitter Library

If you’re going to make use of the EMTwitterMessageController in your app, you will need to follow the attribution requirements, specified in the Twitter+OAuth library license:

```
please include the following text somewhere in your application's user-facing
text:
"Includes Twitter+OAuth code by Ben Gottlieb"
```

Since this library includes code from the MGTwitterEngine library, you should also add:
“Includes MGTwitterEngine code by Matt Gemmell.”

Changing Branding

1. Replace Resources/images/Icon.png with a custom desktop icon.

   This image should be a 57x57 px PNG image. This image can be named differently, as long as the "Icon file" value is set to match in Resources/ecommerce-Info.plist

2. Replace Resources/Default.png with a custom splash screen.

   This image should be a 320x480 px PNG image. This image can only be called Default.png

3.

Supporting Localization

1. Edit the labels on the Right side of the equals sign in Resources/en.lproj/Localizable.strings and Resources/emlib.bundle/en.lproj/Localizable.strings

Changing Data Sources

The concept of a data source is used throughout the iPhone app, as an interface layer between the UI controllers and views and the data that powers these controls. Data sources are typically specific to one kind of UI component and are meant to be easy to swap out to change where the component gets data. For instance, for search typeahead, there are two default data source implementations. The EMDynamicKeywordsDataSource makes dynamic http queries to a backend server to power the results.

Most data sources are configurable in Classes/Configuration/GlobalConfiguration.m

Changing the Search Typeahead Data Source

When changing the Endeca data source, you'll also need to update the Typeahead Data Source to match. The typeahead endpoint is configured in `GlobalConfiguration.m`, and should return an NSString with a "%@" token where the search terms should be filled in.

```
- (NSString *)typeaheadEndpoint {
      return
@"http://example.com/mobile/search.getSuggestions.json?searchTerms=%@";
}
```

This should return a JSON array, where the first item is the search term, and the second item is an array of search suggestions. For example:

["foo", ["food", "fool", "footwear", "football", ...]]

Changing the Product Detail Data Source

The Endeca Camera Store detail API sends back data from an Endeca record detail query served by the Endeca Mobile API. However, since the detail data needed for the product detail page might not be stored in Endeca, it's possible that the detail API is also served from a different server than the Endeca Mobile API. To change the endpoint, override the detailEndpointForProduct: method in `GlobalConfiguration.m`.

```
- (NSString *)detailEndpointForRecord:(EMRecord *)record {
```

```
    return [NSString
stringWithFormat:@"http://example.com/mobile/detail.json?R=%@",
                                item.spec];
}
```

The detail data source should return a JSON dictionary, and can be nested.

Changing the Product Reviews Endpoint

The Endeca Camera Store reviews API serves data that is stored in Endeca under a different record type. In other cases, the review data might come from Power Reviews, Bazaar Voice, or some other system. In these cases, you'll need to use a endpoint, which is specified in `GlobalConfiguration.m`.

```
- (NSString *)reviewsEndpoint {
    return  @"http://example.com/mobile/reviews/api.json";
}
```

Changing the Cart and Checkout Data Source

The Endeca Mobile API comes with a mock data source that can be used as a testing harness. However, this will need to be changed to an API that connects to your backend commerce database for production.

```
- (NSString *)cartEndpoint {
      return @"http://example.com/mobile/cart";
}
```

Changing the Brands Data Source

If you're using the default EndecaMobileBrandController.java in the Mobile API, you will not need to change the Brands Data Source. If you need to change it, you can override the brandsEndpoint method in GlobalConfiguration.m

```
- (NSString *)brandsEndpoint {
      return @"http://example.com/mobile/brands.getList.json";
}
```

Implementing Analytics

To add analytics tracking to the Oracle Endeca iPhone app, you add the following line to AppDelegate_iPhone.m at the top of the applicationDidFinishLaunching: method, where myProvider is a class that implements the EMAnalyticsProvider protocol.

```
[EMAnalytics setProvider:myProvider];
```

The provider class can extend EMAnalyticsProvider and selectively override only the methods that you want to track. The default method implementations all do nothing.

Adding a new tab

If you want to add a new ViewController to the tab bar, you need to edit AppDeletage_iPhone.m. To add a ViewController to the More tab, you need to edit MoreController.m. In either case, if you want the ViewController to be encoded in the app state so that it can load back up if the user exits the app and then enters again, you need to implement the following methods in your ViewController subclass.

```
- (void) encodeWithCoder:(NSCoder*)encoder {
      [super encodeWithCoder:encoder];
      [encoder encodeObject:self.title forKey:@"title"];
}

- (id) initWithCoder:(NSCoder*)decoder {
      if(self = [super initWithCoder:decoder]) {
            self.title = [decoder decodeObjectForKey:@"title"];
      }
      return self;
}

- (BOOL) shouldBeEncodedInAppState {
      return YES;
}
```

Querying Endeca

There are 4 main classes used in querying the Endeca Mobile API: EMSearchQuery, EMSearchResult, EMURLSearchRequest, and EMURLSearchResponse. The EMSearchQuery and EMSearchResult objects are basically just models holding the configuration and data respectively. The EMSearchQuery object encapsulates the Endeca state, such as search terms, breadcrumbs, range filters, sort and paging parameters etc. The EMSearchResult encapsulates the records, guided navigation, and content item. The EMURLSearchRequest and EMURLSearchResponse handle creating the appropriate URL, sending the HTTP request, and parsing the response.

The http request, and response parsing are threaded methods performed asynchronously, with a delegate requestDidFinishLoad method being called upon successful completion. See the example below for how a basic request can be made using these 4 classes.

```
@implementation SampleClass

- (void)sampleMethod {

    EMSearchQuery *query = [EMSearchQuery queryWithRecordType:[EMProduct
class]];
    EMURLSearchRequest *request = [EMURLEndecaSearchRequest
requestWithSearchQuery:query

      endpoint:@"http://example.com/mobile/search/api.json"
                                    delegate:self];
   [request send];
}


- (void)requestDidFinishLoad:(TTURLRequest*)request {

    id<EMURLSearchResponse> response =
(id<EMURLSearchResponse>)request.response;
    EMSearchResult *result = response.result;

}


@end
```