Oracle FLEXCUBE Direct Banking

iOS Client developer Guide
Release 12.0.3.0.0

Part No. E52543-01

April 2014

ORACLE®

iOS Client Developer Guide
April 2014

Oracle Financial Services Software Limited
Oracle Park
Off Western Express Highway
Goregaon (East)
Mumbai, Maharashtra 400 063
India
Worldwide Inquiries:
Phone: +91 22 6718 3000
Fax:+91 22 6718 3001
www.oracle.com/financialservices/

Contents

1. Preface

1.1 Intended Audience

Any interested party working on the delivery of Oracle FLEXCUBE Direct Banking may read this document. The following profile of users would find this document useful:

- Application Architects
- End to End Designers
- Business Service Detailed Designers and Developers
- Implementation Partners

Specifically, however, this document is targeted at Implementation Partners, Customization Development Teams or Vendors providing customization, configuration and implementation services around the Oracle FLEXCUBE Direct Banking product.

1.2 Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

1.3 Access to OFSS Support

https://support.us.oracle.com

1.4 Structure

This manual is organized into the following chapters:

Section 1 *Preface* gives information on the intended audience. It also lists the various chapters covered in this User Manual.

Section 2-4 *Provides introduction and abbreviations details.*

Section 5 *Provides information for creating Xcode setup for iOS development.*

Section 6 *Provides information about components involved in application.*

Section 7-17 *Brief description about every component.*

Section 18 *Provides information to export ".ipa file".*

Section 19 *Explains limitations about naming conventions of UI resources.*

1.5 Related Information Sources

For more information please, refer to the following documents:

- *Oracle_FLEXCUBE_Direct_Banking_Mobile_Android_Client_Developer_Guide*
- *Oracle_FLEXCUBE_Direct_Banking_Mobile_J2ME_Clients_Developer_Guide*
- *Oracle_FLEXCUBE_Direct_Banking_Mobile_Banking_User_Interface_Guide*

2. Introduction

FLEXCUBE Direct Banking mobile client applications are supposed to be designed in order to understand the XML structure and render it on the mobile device screen appropriately. This stands as the basic requirement for iOS client as well. The iOS mobile client is a client application specifically targeted for Apple's iOS platform. This document is a generic guide for development of iPad and iPhone client Application.

FLEXCUBE Direct Banking's mLEAP framework is the entity responsible for generating the content for mobile clients. This content is represented in a pre-defined XML format. For details on the XML structure, please refer to the document Oracle_FLEXCUBE_Direct_Banking_Mobile_App_XML_structure.docx.

Some key points about developing an application on iOS platform are highlighted below:

- iOS applications(Both iPhone and iPad) are written in the Objective C programming language.
- The compiled Objective C code along with Custom Frameworks and resource files required by the application is bundled by the iOS Asset Packaging Tool into an iOS package, an archive file marked by an .ipa suffix.
- This file is the vehicle for distributing the application and installing it on Apple mobile devices; it's the file users download to their devices.
- All the code in a single ipa file is considered to be one application.
- The ipa file is signed with a certificate before deployment.
- The client receives the response from the server in XML and parses it to render the screen.
- The screen rendering logic is written in client who majorly uses the native components of the iOS SDK framework.

3. Acronyms and Abbreviations

FCDB	FLEXCUBE Direct Banking
IDE	Integrated Development Environment
iOS	Apple OS for iPhone/iPod Touch/iPad
SDK	Software Development Kit
UI	User Interface
IPA	iOS App Store Package

4. Scope

This document speaks about various design and development level considerations for the FLEXCUBE Direct Banking mobile client for iOS platform. It describes the high level design, process flow, various key components involved, screen layout design etc. This mobile client is targeted at Apple's iOS platform. Some of the terms used in this document are specific to the iOS platform. This document does not go in detail of the iOS platform features but speaks only with relevance to the FLEXCUBE Direct Banking mobile client architecture. Reference sites for iOS Development are mentioned wherever applicable.

5. Xcode Setup

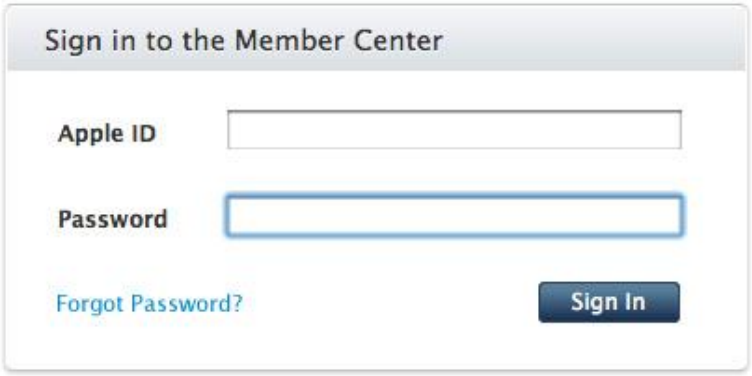
5.1 iOS SDK and Xcode IDE

Xcode is a developer toolkit for Mac, iPhone, and iPad. It includes the Xcode IDE, iOS Simulator, and all required tools and frameworks for building Mac OS X and iOS applications.Xcode has unified user interface design, coding, testing, and debugging all within a single window. The Xcode IDE analyzes the details of your project to identify mistakes in both syntax and logic, it can even help fix your code for you. Before you can install or use Xcode IDE, you must have a compatible version of OS X installed on your Mac Machine. Refer *Oracle_FLEXCUBE_Direct_Baking_Mobile_iPhone_Workspace_Configuration.docx* for information on version compatibility.

5.2 Downloading the SDK Starter Package

This guide describes Xcode 5. If you're using Xcode 4, read *App Distribution Guide for Xcode 4 click* https://developer.apple.com/legacy/library/navigation/

You can download it from the Apple Developer Center*, though you have to be registered as an apple developer. Registration is free and relatively painless.*



Once you are logged in, select the link in the for Mac Dev Center

Download the latest version of Xcode 4 or Xcode 5.
(Refer Oracle_FLEXCUBE_Direct_Baking_Mobile_iPhone_Workspace_Configuration.docx for information on version compatibility.)

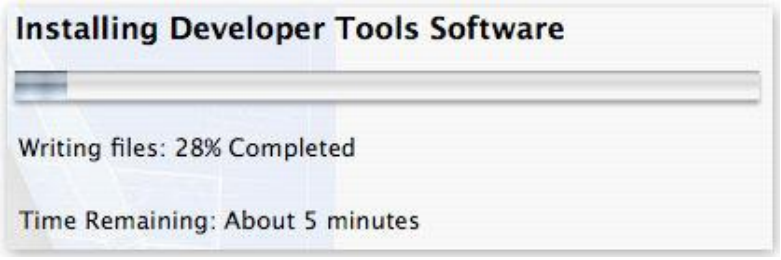
5.3 Installing Xcode

Once the download completes, mount the installer and run the XcodeTools package. Click through the installer till you see the customize option.

Click it. This gives you the option to skip certain parts of the install. If you are short on disk space you'll probably want to skip the 1.3GB of developer documentation.

It is also recommended that you check any boxes that indicate support for older versions of OS X ie 10.5 / 10.6 etc.

The installer starts installing.



Once installation is complete, Drag it to the applications folder of your Mac.

The Developer Tools installer will have installed a folder called Developer in the root (highest level) of your hard drive. The Xcode application is located at /Developer/Applications/Xcode.app

Xcode has been successfully installed on your computer.You can go to Finder > applications > xcode .Click and open it.

5.6 Running FCDB Mobile banking Project

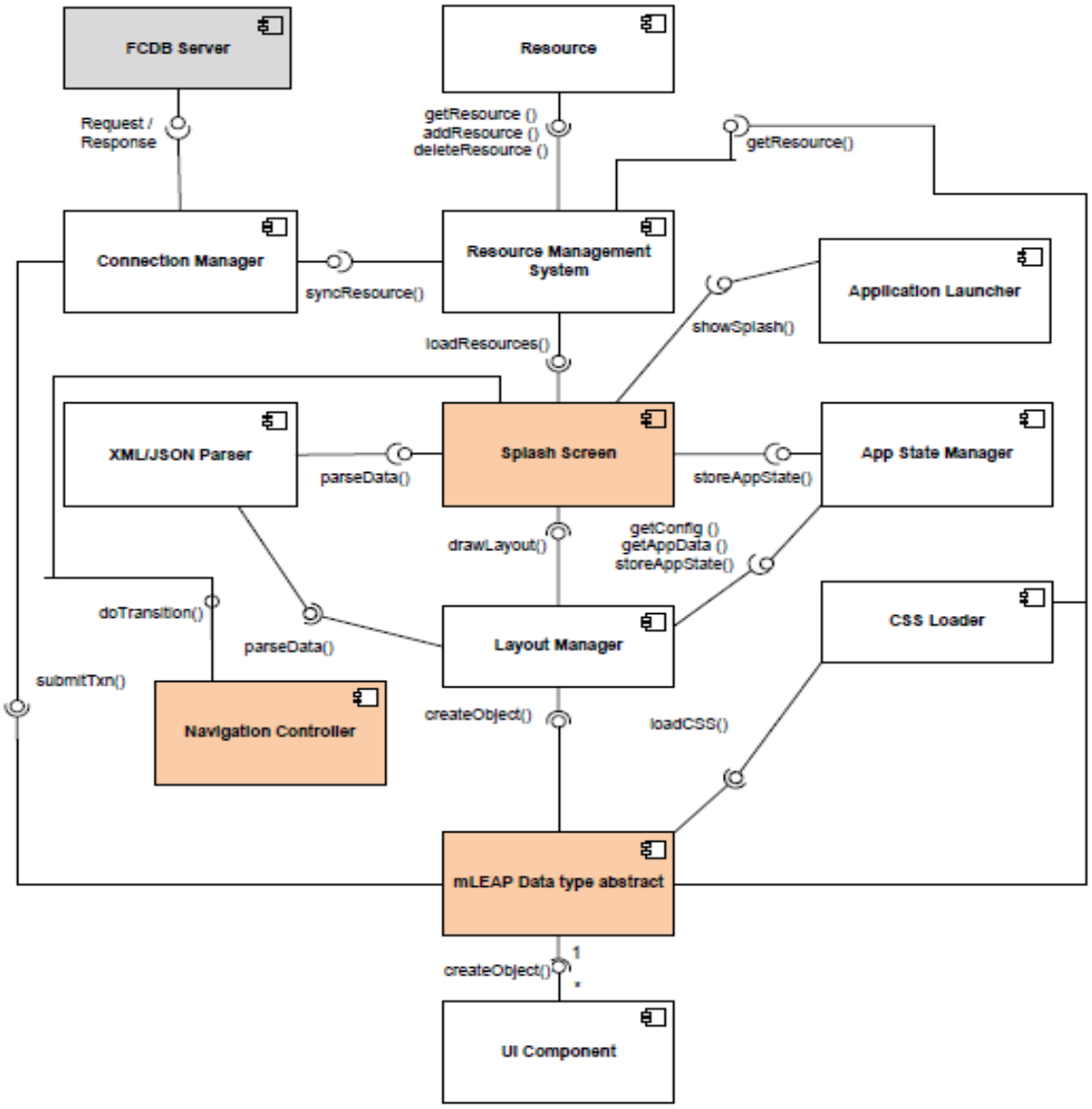
Refer Oracle_FLEXCUBE_Direct_Baking_Mobile_iPhone_Workspace_Configuration.docx and follow step by step to configure application

5.7 Running Simulator

- Build and Run the application with target as simulator. On successful build, you should find the Simulator presenting the app launch screen. You can select iPad simulator or iPhone simulator for iPad and iPhone application respectively.

6. System Design

6.1 Component Diagram



Here name of the components and methods are indicative. It is not necessary that same package or method name will be available in application. They are just for reference.

6.2 Roles of the components involved

- **FCDB Server:** This is the external entity to the mobile app to which the client app communicates in the form of request and response.

- **Application Launcher:** The first point which is triggered when the application is launched.

- **Splash screen:** There will be a configurable splash screen. It will be triggered by Application Delegate component.

- **Connection Manager:** This is primarily involved in establishing the connection with FCDB Setup and passing the request XML over HTTP/HTTPS.

- **Resource Management System:** The main role of this component is to sync the resources which are being used at the client side with the latest versions present or updated in case at the server side.

- **Resource:** These are any resources which are present at the client side and can include images,css files for styling, language pack and client side xml.

- **Layout Manager:** It is responsible for managing the client side layout.

- **App State Manager:** This component is responsible for managing the overall state of the application.

- **CSS Loader:** The primary responsibility of this component is to load the CSS and to observe any theme changes and refresh the style.

- **mLEAP Data-type abstract:** It will trigger the creation of all the primitive and complex UI components which in turn would be styled and customized using the CSS and finally instantiated.

- **UI Component:** The primitive as well as complex UI components which are finally rendered on the screen by the composite data type.

7. FCDB Server

This is the external entity to the mobile app to which the client app communicates in the form of request and response by the help of usability xslts. It sends request(for mobile app it is request whereas for transaction point of view it is a response) in the form of F-XMLs. After parsing these XMLs and using their attributes and elements we paint the specified screens. After submitting from the screen the data again goes back to FCDB Server.

8. Application Delegate

The first point which is triggered when the application is launched. It is a default behavior of Xcode. The Application Delegate will also trigger the configured splash screen for display. It will trigger the Resource Management System to check if there is any update on the server side in terms of any resource updates etc. If yes it will be handled by the Resource Management System.

9. Splash Screen

This will load properties from GlobalConfigurationFile.plist. The associated file with this component is named as SplashViewController. It provides hook to the login screen of the application.

10. Connection Manager

A custom framework -FCDBConnectionManager.framework is used in our application as connection manager which is responsible for establishing connection between FCDB Server and iOS mobile App.

- We read URL parameters of FCDB server from GlobalConfigurationFile.plist.
- All app level configurations like language Id, device Id, user agent is stored in GlobalDictionary.plist and is read here.
- We also read values corresponding to the keys mentioned in GlobalKeys.plist and use this information to construct request .It includes fldEncrKey, idsession, etc.
- It is designed to handle JSON and XML both.

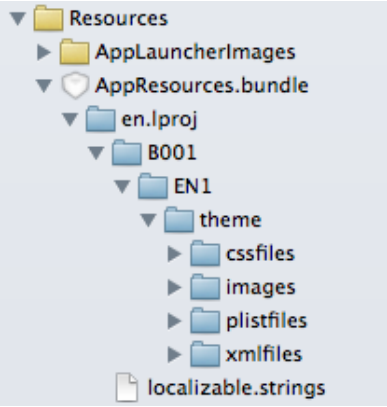
11. Resource Management System

A custom framework - FCRMS.framework is used in our application as Resource Management System.Suppose if you want to change a specific image or style in the mobile app but do not want to force user to download the complete app for this small change. In this type of scenario, resource management system comes to resort by providing a means by which we can simply upload the image or style from the FCDB server and RMS will ensure that user will get the latest image or style in the app without having them download the application again. This system is initialized from AppDelegate.m.

Other use of RMS frameworks are classified as-

- Depending upon the device I.e iPad or iPhone loads the corresponding images,client side xml and css files
- handles application orientation
- update Selected Theme Dictionary
- stores the list of supported language and provides the flexibility to localize the application based on language setting
- handles both right to left and left to right alignment.

12. Resources



It typically contains AppResources.bundle which has folder based on language.For example: en.lproj corresponds to english language.Please note this is the default folder and all the other language folder will contain just the incremental changes.

All application images, CSS files, layout XMLs ,plist files can be located in their respective folders. Localizable.strings file contains the localized value for strings present client side.The Resource Management System ensures the up-to-date version of these resources by keeping them in sync with the server.You will find a similar structure for other language as well.

You can use FCRMSKeys.plist to change the default language of the application and preferred theme for the application.Properties like Application version, application URL can be accessed and modified using this file.

Please change the certificate file named *fcdb_sign.cer*. This contains the public key and is used for encryption during login process. It can be located inside Resources>others.

13. Layout Manager

It handles the different mLEAP data-types and primitive components which contribute for a screen display like all form elements etc. Mainly this component includes :-

- ErrorHandler.m
- BaseXMLContentGenerator.m

13.1 ErrorHandler.m

This class is the entry point for the transactions. We filter out the errors here and display them in custom dialog. If no error is found we redirect the flow to the performActionOnScreenInfo method of BaseXMLContentGenerator class to handle further activity.

13.1 BaseXMLContentGenerator.m

This class is responsible for creating layouts on the basis of parsed F-XML.Major Methods in this class are-

- (void)flushAllPreProcessDictionary; It flushes all the previous dictionary for the current request id.

- (void)preProcessTheScreenInfo; pre process the response received from the server. It calls a loop on all the mLEAP datatypes that occurs and create a views and add them in the main layout.The method call is recursive. It also invokes postDrawingProcess method after completion of addition of all views in main layout. Based on actid it invokes xmlTestContentController or xmlVerifyViewController or LookUpTableXMLGenerator.

- (void)postDrawingProcess; After completion of the UI process, We use this method to fire first time behaviour and displaying modal windows.After which it flushes the behavior dictionary.

- (void)handleBehaviourArray:(NSArray *)behaviours; This method is invoked from mleap datatypes and postDrawingProcess method.It aims at achieving functionalities such as show/hide of different views, setting a value in specific view or filtering values of drop down(spinner) or list and may more. It contains methods which are used to serve specific functionalities.

14. Application state Manager

- It is a singleton class which is used to hold all the global level variables that is to be held throughout the application session.
- It store application state- Logged Out, First Login,etc
- It store user login information-name, last login, current login, time zone, current request id
- It stores menu information and favorite information. It also extra menu items configurations which needs to invoke a client controller.
- Holds the widget content controllers Dictionary
- It initializes the dictionary for current request and deletes all previous value stored.

15. CSSLoader

CSSLoader component includes mainly three files named as

- FCStylesheetCache.m
- FCCSSRuleset.m
- Style.css

15.1 FCStylesheetCache.m

This class is used to load specified css file through RMS. Its applyStyleWithRuleSet method returns the UI for the particular mLEAP datatype after application properties corresponding to specified css attributes. This class is invoked from AppDelegate.m.

15.2 FCCSSRuleset.m

This class contains various attributes which could be modified in order to style any UI.Every attribute has two major methods associated to it .

Example: For KTitleColor=@"title-color";
- (BOOL)hasTitleColor : To check if title-color property is present for any given UI Component.It return a boolean value

- (UIColor *)titleColor : It return the title color if present.

15.3 Style.css

This file contains all the class and its attributes which in turn defines the style guide for various UI components

Note: Attribute names are case sensitive.

16. mLEAP DataType

For each mLEAP Data type which is being used there will be corresponding implementation class in this component. Important files are:-

- FCDBModelObjectDictionary.plist
- TagModelBaseClass

16.1 FCDBModelObjectDictionary.plist

It keeps the corresponding class name that needs to be called for a given mLEAP Tag.This logic is placed in TagModelUnmarshaller class of Parser framework.

16.2 TagModelBaseClass.m

This is a abstract class which has many abstract methods ,of which the most important ones are-

- (id)initWithInfoDictionary:(NSDictionary *)infoDictionary;
- (void)analyseTheModelForActions;
- (UIView *)returnViewForModelForParentViewRect:(CGRect)parentViewRect forDomInfo:(FCDOM *)dom;

This class must be extended by all the mleap datatypes. Description of some important methods are as follows:-

- **(id)initWithInfoDictionary:(NSDictionary *)infoDictionary;** This method is used to create the objects of mLEAP datatype and it returns the same object. In this method we normally get the data from the F-XML attributes and store them.

- **(UIView *)returnViewForModelForParentViewRect:(CGRect)parentViewRect forDomInfo:(FCDOM *)dom;** This method is used to construct the view for the particular mleap datatype. We create UI component here, by which the actual layout would be drawn on the screen. We use to handle listeners, for that UI component, in this method. Also we set the id for view in this method.

Note:-While creating new mLEAP datatype, we need to map respective node of F-XML with the new className of that datatype in FCDBModelObjectDictionary.plist and implement it in parser class.It is working like a reflection of the Tag and you can directly change it using the plist file.

17. mLEAP DataType Category

These are the category written over the mLEAP DATAType model class.This gives us the flexibility to define which View class needs to be called for rendering the view for a Tag.
Two Main methods that needs to be implemented are-

-(void)analyseTheModelForActions :- Incase some data associated with mLEAP Data-types need to be processed and its values needs to be stored

-(UIView *)returnViewForModelForParentViewRect:(CGRect)parentViewRect forDomInfo:(FCDOM *)dom :- It contains the logic for defining frame with respect to the parent rect and calling custom view class and styling the same using dom object.

18. UI Components

These are the components actually displayed on the screen. For creating these classes we need to extend the iOS API components such as UIButton, UITextfield, UITableview, etc.We have created few custom views as well to handle some complex view.All the view are inherited from BaseCustomTagView. This contains default behavior like

- (void)hideView
- (void)showView
- (void)enableView
- (void)disableView
- (void)clearsFldObjects
- (void)increaseHeight
- (void)decreaseHeight

We are doing two main task inside every view created:-

- Defining frame and view for the abstract data Type
- Applying style on view with CSSAttributes

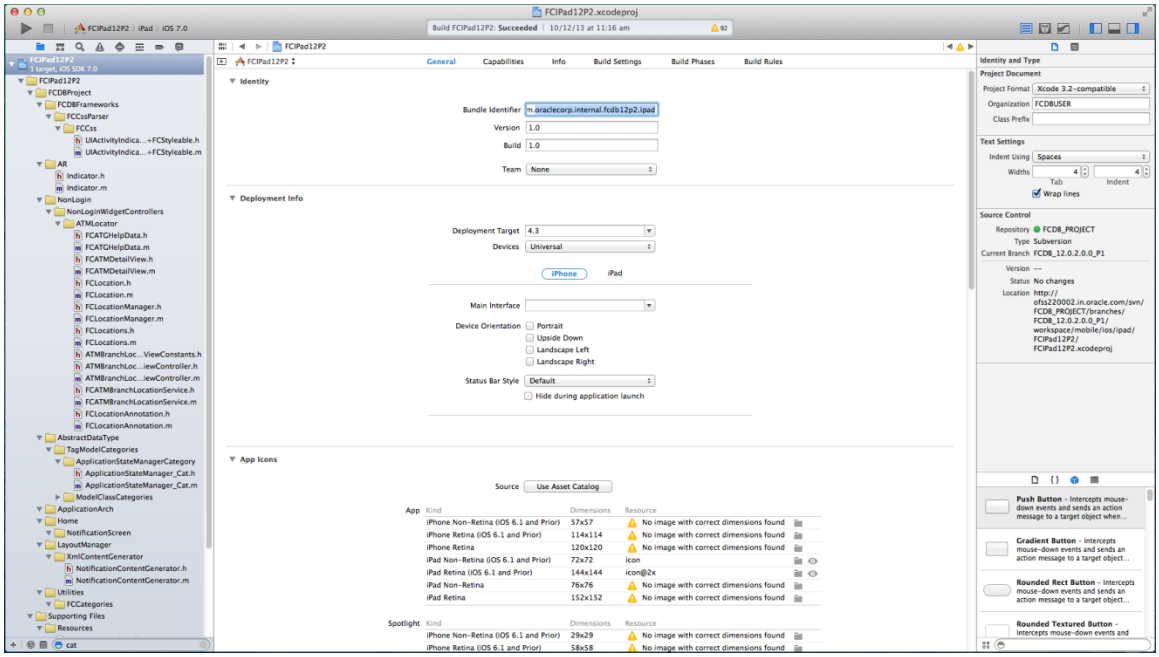
Also we need to define the view incase of error.All this is done here.

19. Exporting Final Output IPA File

19.1 Export Mechanism

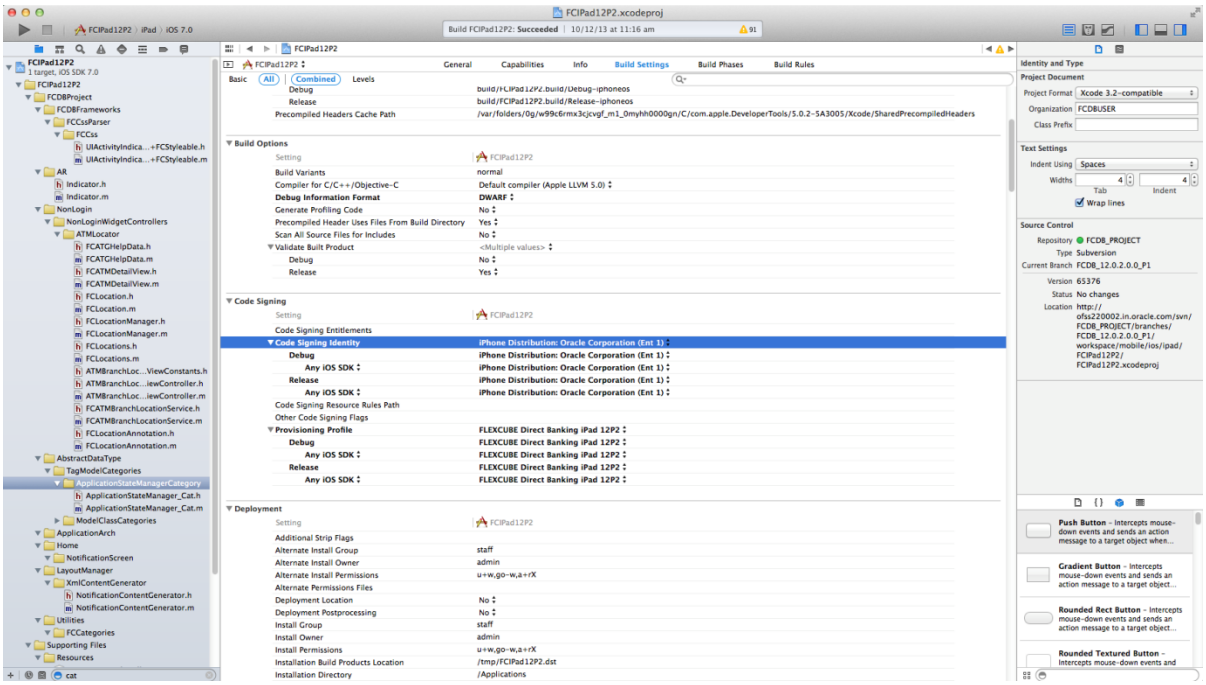
NOTE: You will need an Apple Developer or Enterprise account to prepare an .IPA to be distributed. You will also need distribution provisioning profile or Ad Hoc Profile

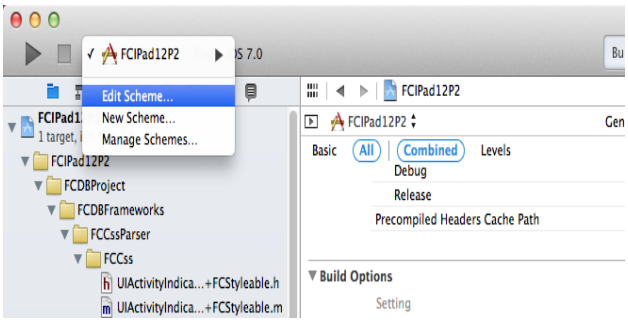
- In the file list on the left drop down menu, select your target. On the Summary tab, notice the



Bundle Identifier is correct.

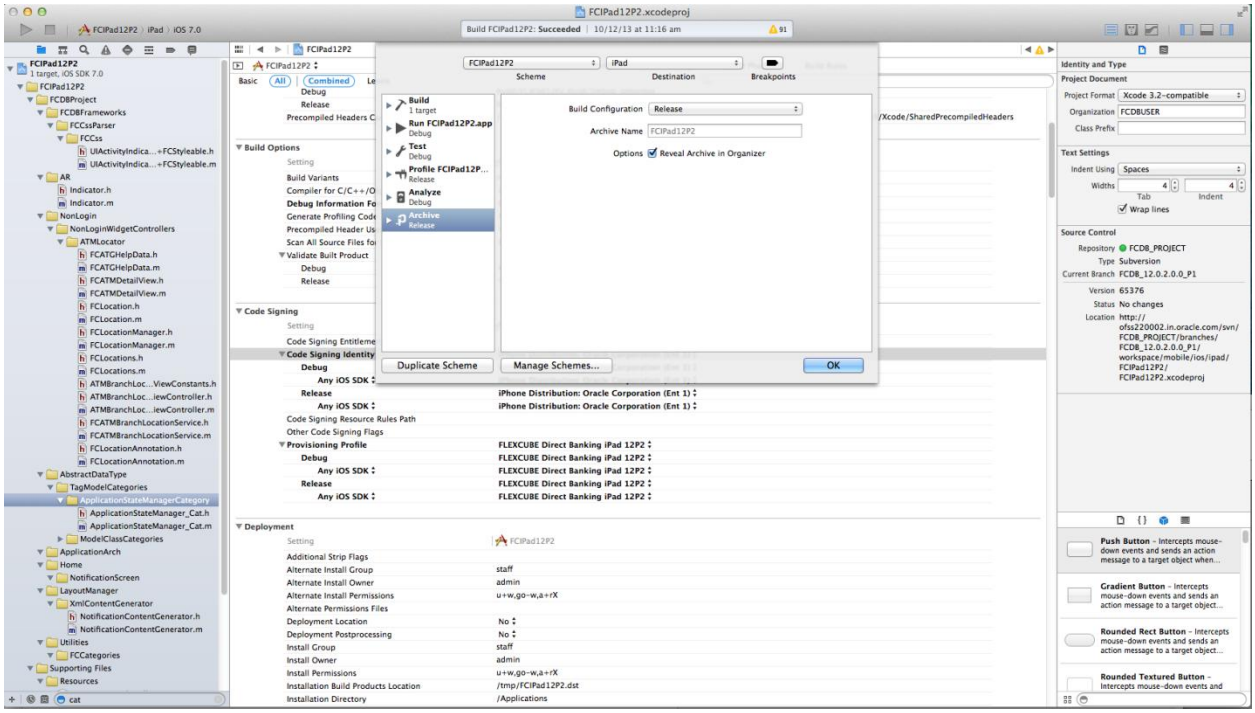
- Go to the Build Settings tab and scroll down to the Code Signing section and expand the Provisioning Profile settings. For the two default build configurations (Debug and Release), select your Development Provisioning Profile for Debug and your Ad Hoc Provisioning Profile for Release.





- In the dropdown in the top left, select Edit Scheme. You will notice that the default build configurations have already been configured based on the build action being performed. If you are performing a Run action (Product -> Run) to test your application on the iOS Simulator or on your device, your app will be built using the Debug configuration, which in turn signs your app with the Development Provisioning Profile. If you are performing an Archive action (Product -> Archive), your app will

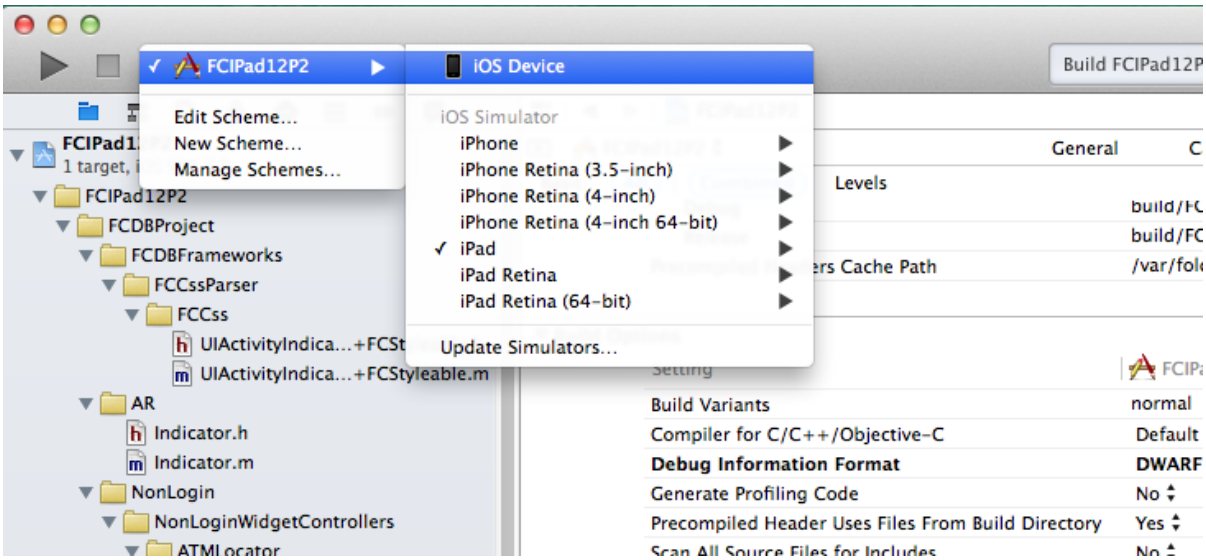
be built with the Release configuration, which in turn signs your app with your Distribution Provisioning Profile (either Ad Hoc or App Store).



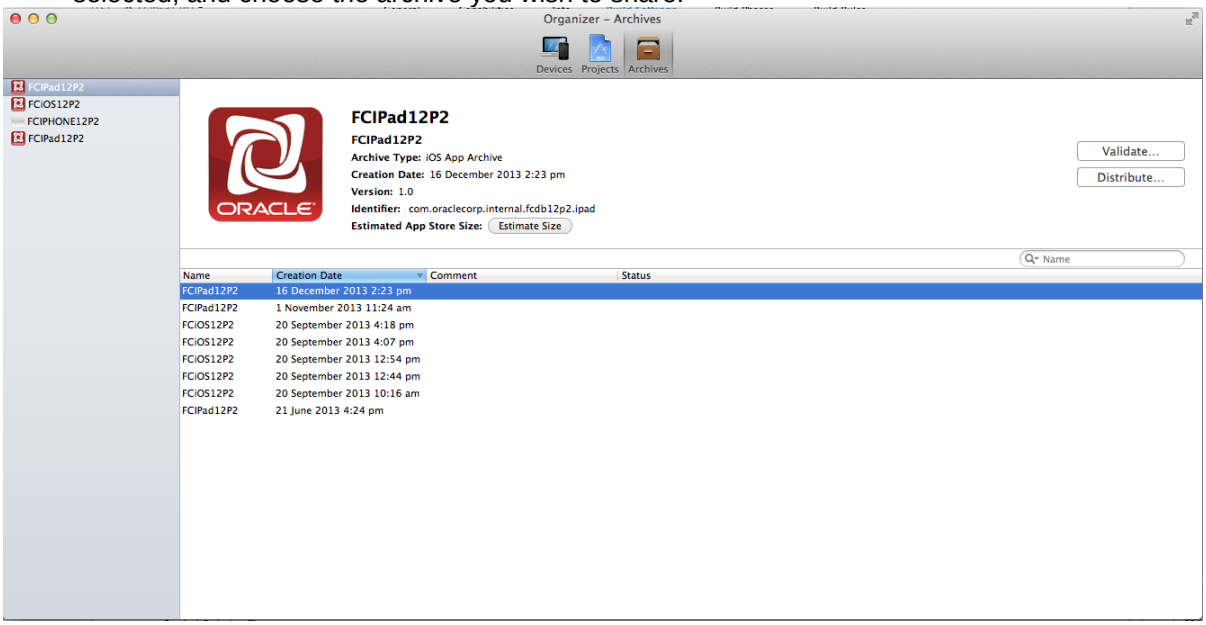
19.2 Generate the IPA

The easiest way to generate an IPA is to archive your applications and share it from the Xcode Organizer.

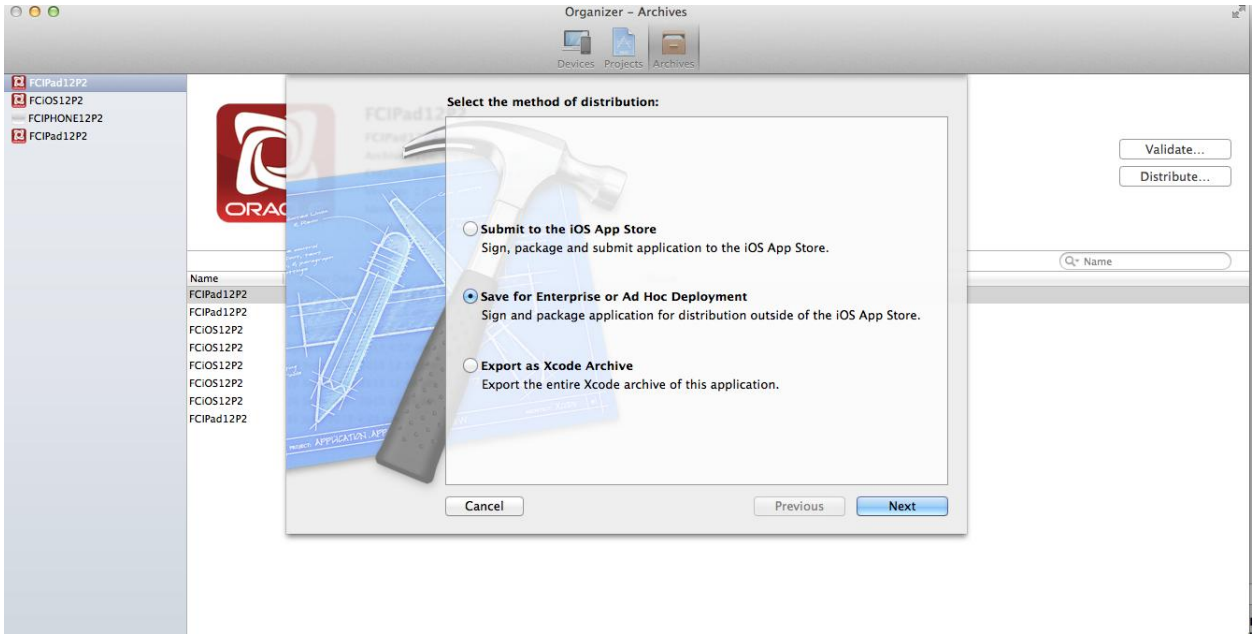
- Change the build target from iPad/iPhone Simulator to iOS Device.



- Under the Product menu, select Archive. This will build your application and code sign it using the Distribution (Ad Hoc or App Store) profile Once the build has completed the Organizer window will appear. If it does not, open it using CMD-Shift–2 or Window -> Organizer. If you see a message popup saying “code sign wants to sign using key ”privateKey“ in your keychain.”, select Allow or Always Allow.
- Go to the Archives tab in Organizer and select your application, if it was not automatically selected, and choose the archive you wish to share.



- Click the Distribute button. In the next window select “Save for Enterprise or Ad-Hoc Deployment”, and click Next.



- In the Code Signing Identity drop down, select the same Distribution Provisioning Profile specified in the Release configuration that you have used during setting up of application, and click Next. **NOTE: When generating an IPA for distribution on FCDBiPhone or FCDBiPad, you should always use an Ad Hoc Distribution Provisioning Profile for both the Archive and Distribute options.**
- Select where you would like to save your IPA and upload to FCDBiPad.ipa /FCDBiPhone.ipa .

20. UI Resource Guide

Please keep in mind below limitations on the naming conventions of images:

- The name of two image files (excluding extension) cannot be same
- Image Folder name should not be in small case.

Please keep in mind below limitations on the naming conventions of CSS file:

- Class name should not contain "_".
- Class name and property name are case sensitive.

For more information please refer following document:-

Oracle_FLEXCUBE_Direct_Banking_Mobile_Banking_User_Interface_Guide.pdf