Tutorial: iOS Object API Application Development

SAP Mobile Platform 2.3 SP04

DOCUMENT ID: DC01938-01-0234-01
LAST REVISED: March 2014

Sybase, Inc., One Sybase Drive, Dublin, CA 94568.

Contents

SAP Mobile Platform Tutorials

The SAP® tutorials demonstrate how to develop, deploy, and test mobile business objects, device applications, online mobile applications (native OData and REST services based), and Hybrid App packages. You can also use the tutorials to demonstrate system functionality and train users.

- Learn mobile business object (MBO) basics, and use this tutorial as a foundation for the Object API application development tutorials:
  - *Tutorial: Mobile Business Object Development*

    Note: For all Object API tutorials, if you opt to use the Mobile Business Object example project instead of performing the Mobile Business Object Tutorial, you must deploy the mobile application project to SAP Mobile Server as a prerequisite.
- Create native Object API mobile device applications:
  - *Tutorial: Android Object API Application Development*
  - *Tutorial: BlackBerry Object API Application Development*
  - *Tutorial: iOS Object API Application Development*
  - *Tutorial: Windows Object API Application Development*
  - *Tutorial: Windows Mobile Object API Application Development*
- Create a mobile business object, then develop a hybrid app package that uses it:
  - *Tutorial: Hybrid App Package Development*
- Create an OData mobile application with REST Services
  - *Tutorial: Android OData Application Development with REST Services*
  - *Tutorial: iOS OData Application Development with REST Services*

Getting Started with SAP Mobile Platform (On-Premise)

Install and learn about SAP Mobile Platform and its associated components.

Complete the following tasks for all tutorials, but you need to perform them only once.

1. *Installing SAP Mobile Platform*

   Install SAP Mobile SDK and SAP Mobile Platform Runtime.
2. *Starting SAP Mobile Platform Services*

   Start SAP Mobile Server, SAP Control Center, the sample database, the cache database (CDB), and other essential services.
3. *Starting SAP Mobile WorkSpace*

   Start the development environment, where you can create mobile business objects (MBOs), create connection profiles and manage SAP Mobile Server connections, develop Hybrid Apps, and generate Object API code.
4. *Connecting to SAP Control Center*

   Open SAP Control Center to manage SAP Mobile Server and its components.
5. *Learning SAP Mobile WorkSpace Basics*

   SAP Mobile WorkSpace features are well integrated in the Eclipse IDE. If you are unfamiliar with Eclipse, you can quickly learn the basic layout of SAP Mobile WorkSpace and the location of online help.

Installing SAP Mobile Platform

Install SAP Mobile SDK and SAP Mobile Platform Runtime.

Before starting this tutorial, install all the requisite SAP Mobile Platform components. See the SAP Mobile Platform documentation at *http://sybooks.sybase.com/sybooks/sybooks.xhtml?id=1289&c=firsttab&a=0&p=categories*:

- *Release Bulletin*
- *Installation Guide for SAP Mobile SDK*
- *Installation Guide for Runtime*

1. Install these SAP Mobile Platform Runtime components:
   - Data Tier (included with single-server installation)
   - SAP Mobile Server
2. Install SAP Mobile SDK, which includes:

- Development support for native Object API and OData SDK applications, as well as HTML5/JS Hybrid Apps.
- SAP Mobile WorkSpace, the Eclipse-based development environment for MBOs and Hybrid Apps.

Starting SAP Mobile Platform Services

Start SAP Mobile Server, SAP Control Center, the sample database, the cache database (CDB), and other essential services.

The way in which you start SAP Mobile Platform Services depends on the options you selected during installation. You may need to manually start SAP Mobile Platform Services. Select **Start > (All) Programs > SAP > Mobile Platform > Start SAP Mobile Platform Services**.
The following services will be started:

- SAP Control Center *<Version>*
- SAP Mobile Platform Cache DB
- SAP Mobile Platform SampleDB
- SAP Mobile Server

SAP Mobile Platform Services enable you to access the SAP Mobile Platform runtime components and resources.

Note: The SAP Mobile Platform installer creates the Windows service (SAP Mobile Platform Sample DB) that runs the sampledb server only when you install SAP Mobile Server with a Personal or Enterprise Development license. If you installed SAP Mobile Server with an Enterprise Server (production) license, you must create this service using the `sampledb.bat` command line utility. See *Create or Remove the Windows Service for sampledb Server (sampledb) Utility* in *System Administration* for more information about using this command line utility.

Starting SAP Mobile WorkSpace

Start the development environment, where you can create mobile business objects (MBOs), create connection profiles and manage SAP Mobile Server connections, develop Hybrid Apps, and generate Object API code.
Select **Start > (All) Programs > SAP > Mobile Platform > Mobile WorkSpace 2.3**.
The SAP Mobile WorkSpace opens in the Mobile Development perspective. The Welcome page displays links to the product and information.

Next

To read more about SAP Mobile WorkSpace concepts and tasks, select **Help > Help Contents**.

Connecting to SAP Control Center

Open SAP Control Center to manage SAP Mobile Server and its components.

From SAP Control Center, you can:

- View servers and their status
- Start and stop a server
- View server logs
- Deploy a mobile application package
- Register application connections
- Set role mappings
- Assign/Unassign a hybrid application to a device

For information on configuring, managing, and monitoring SAP Mobile Server, click **Help** > **Help Contents**.

1. Select **Start** > **(All) Programs** > **SAP** > **SAP Control Center**.

   Note: If SAP Control Center does not launch, make sure that the SAP Control Center service is started in the Windows Services dialog.

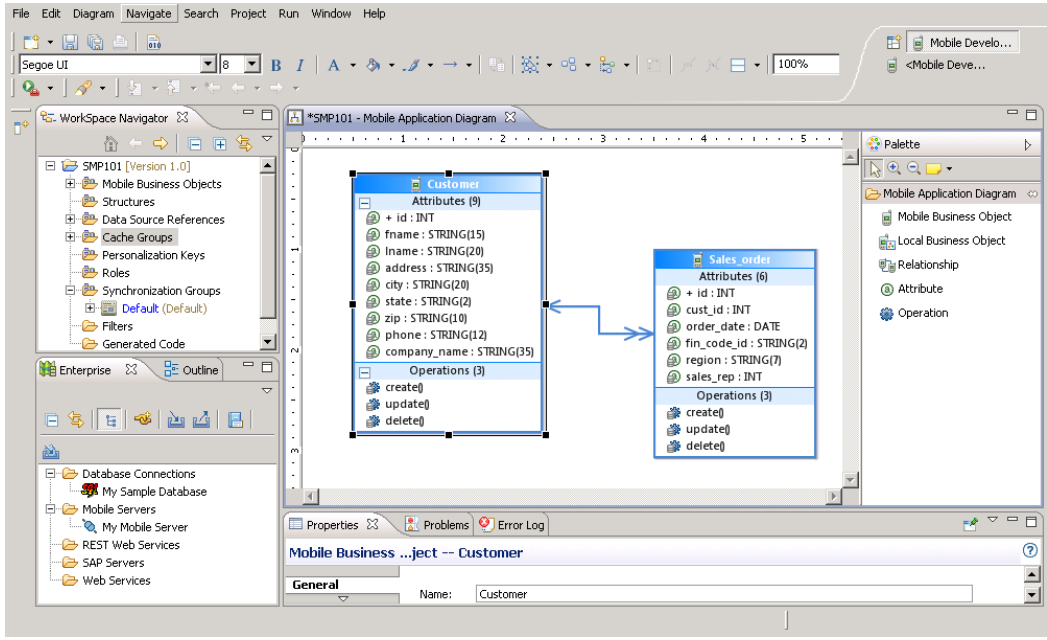
2. Log in by entering the credentials set during installation.

   SAP Control Center gives you access to the SAP Mobile Platform administration features that you are authorized to use.

Learning SAP Mobile WorkSpace Basics

SAP Mobile WorkSpace features are well integrated in the Eclipse IDE. If you are unfamiliar with Eclipse, you can quickly learn the basic layout of SAP Mobile WorkSpace and the location of online help.

- To access the online help, select **Help** > **Help Contents**. Some documents are for SAP Mobile WorkSpace, while others are for the Eclipse development environment.
- The Welcome page provides links to useful information to get you started.
  - To close the Welcome page, click **X** in the upper right corner of the page.
  - Reopen the Welcome page by selecting **Help** > **Welcome**.
  - To learn about tasks you must perform, select the **Development Process** icon.
- In SAP Mobile WorkSpace, look at the area (window or view) that you will use to access, create, define, and update mobile business objects (MBOs).



Window	Description
WorkSpace Navigator view	Use this view to create Mobile Application projects, and review and modify MBO-related properties. This view displays mobile application project folders, each of which contains all project-related resources in subfolders, including MBOs, datasource references to which the MBOs are bound, personalization keys, and so on.
Enterprise Explorer view	A view that provides functionality to connect to various enterprise information systems (EIS), such as database servers, SAP® back ends, and SAP Mobile Server.

Window	Description
Mobile Application Diagram	The Mobile Application Diagram is a graphical editor where you create and define mobile business objects. Use the Mobile Application Diagram to create MBOs (including attributes and operations), then define relationships with other MBOs. You can: • Create MBOs in the Mobile Application Diagram using Palette icons and menu selections – either bind or defer binding to a datasource, when creating an MBO. For example, you may want to model your MBOs before creating the datasources to which they bind. This MBO development method is sometimes referred to as the top-down approach. • Drag and drop items from Enterprise Explorer to the Mobile Application Diagram to create the MBO – quickly creates the operations and attributes automatically based on the datasource artifact being dropped on the Mobile Application Diagram. Each new mobile application project generates an associated mobile application diagram.
Palette	The Palette is accessed from the Mobile Application Diagram and provides controls, such as the ability to create MBOs, add attributes and operations, and define relationships, by dragging and dropping the corresponding icon onto the Mobile Application Diagram or existing MBO.
Properties view	Select an object in the Mobile Application Diagram to display and edit its properties in the Properties view. While you cannot create an MBO from the Properties view, most development and configuration is performed here.
Outline view	Displays an outline of the active file and lists structural elements. The contents are editor-specific.

Window	Description
Problems view	Displays validation errors or warnings that you may encounter in addition to errors in the Diagram editor and Properties view. Follow warning and error messages to adjust MBO properties and configurations to avoid problems, and use as a valuable source for collecting troubleshooting information when reporting issues to Customer Service and Support.
Error Log view	Displays error log information. This is a valuable source for collecting troubleshooting information.

Developing an iOS Application

Generate Object API code for the iOS platform, develop a universal iOS device application with code, and test its functionality. The device application communicates with the database MBOs that are deployed to SAP Mobile Server.

Prerequisites

Note: This tutorial has been developed using SAP Mobile Platform 2.3 SP04, Mac OS X 10.8.5, iOS SDK 7.0.3, and Xcode 5.0.1 Development Environment, and executed on an iOS Simulator v 7.0. If you use a different version, some steps may vary. For more information on Xcode, refer to the Apple Developer Connection: *http://developer.apple.com/technologies/tools/whats-new.html*.

1. Complete the tasks in *Getting Started with Mobile Platform*.
2. Either:
   - create the MBO project by completing *Tutorial: Mobile Business Object Development*, or
   - download and deploy the MBO SMP101 example project (complete project files) from the SAP® Community Network: *http://scn.sap.com/docs/DOC-8803*.

   Note: If you upgrade SAP Mobile SDK after completing the tutorial, you can convert the project to the current SDK by importing the earlier project into the SAP Mobile WorkSpace and then accepting the confirmation prompt.
3. (Optional) To use as a reference and copy source code when completing this tutorial, download the iOS SMP 101 example project (source code only) and extract to your Mac from the SAP® Community Network: *http://scn.sap.com/docs/DOC-8803*.

Task

1. *Installing Xcode IDE*

   Download and install the Apple Xcode IDE to build, deploy, and run a mobile application on an iPhone simulator.
2. *Generating Object API Code*

   Launch the code generation wizard and generate the object API code for a replication-based iOS application.
3. *Setting Up an iOS Client Application in Xcode*

   Set up an iOS client application in the Xcode IDE.
4. *Registering the Application Connection in SAP Control Center*

   Register the iPhone simulator in SAP Control Center.

5. *Viewing the CallbackHandler and ApplicationCallbackHandler Files*

   Look at CallBackHandler and ApplicationCallbackHandler files in Xcode.
6. *Creating the User Interface*

   Use Interface Builder to create and configure the user interface for the SMP101 application.
7. *Deploying the Device Application*

   Deploy the SMP101 application to the iPhone simulator for testing.

Installing Xcode IDE

Download and install the Apple Xcode IDE to build, deploy, and run a mobile application on an iPhone simulator.

1. Open the App Store and search for Xcode.
2. Install Xcode, entering a valid Apple ID and password.

Generating Object API Code

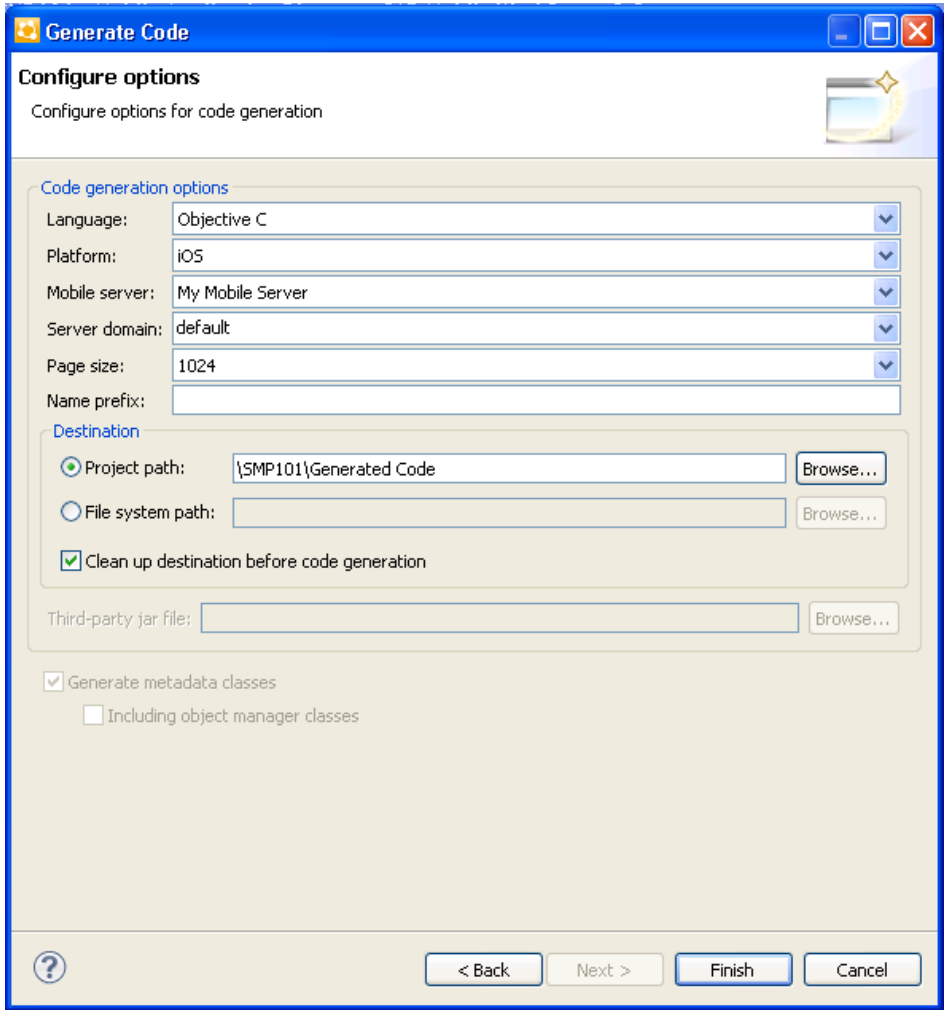
Launch the code generation wizard and generate the object API code for a replication-based iOS application.

1. In SAP Mobile WorkSpace, open the **SMP101** mobile application project.

   In WorkSpace Navigator, right-click the **SMP101** folder and select **Open in Diagram Editor**.
2. Right-click anywhere in the SMP101 - Mobile Application Diagram and select **Generate Code**.
3. On the Code generation configuration screen, click **Next**.
4. Make sure the Customer and Sales_order MBOs are selected, then click **Next**.
5. In the Configure Options page, specify these values and click **Finish**.

Option	Description
Language	Select **Objective C**.
Platform	Accept the default, **iOS**.
Mobile Server	Select **My Mobile Server**.
Server domain	Accept **default**. If you are not connected to SAP Mobile Server, this field is empty. Connect to SAP Control Center to proceed.

Option	Description
Page size	Accept the default or select a larger page size.
Name prefix	The prefix for the generated files. Leave blank.
Project path	Accept the default or enter a different location for the generated project files.
(Optional) Clean up destination before code generation	Delete all items in the destination folder before generating the device client files.



6. Click **OK** in the Success dialog.

Objective-C code is generated into the specified output location and in the WorkSpace Navigator.

Setting Up an iOS Client Application in Xcode

Set up an iOS client application in the Xcode IDE.

Prerequisites

- Generate Objective-C code in to an output location.
- Verify that SAP Mobile Platform is installed in a shared directory so you can access it from your Mac.
- To help create your project, and to later build the interface, download and import the SMP 101 iOS Object API (2.3) example project from the SAP Community Network (SCN) at *http://scn.sap.com/docs/DOC-8803*.
- Copy the SMP101 iOS Object API example project to your Mac machine and extract it into a folder. The example project contains the Xcode project and a SMP101 project archive file to use in SAP Mobile WorkSpace.

Task

1. On your Mac, start Xcode and select **Create a new Xcode project**.
2. Select **iOS Application** and **Empty Application** as the project template, and then click **Next**.
3. Specify these values and click **Next**.
   a) Enter `SMP101` as the product name.
   b) Enter `My Corporation` (or another value as needed) as the organization name.
   c) Enter `MyCorp` (or another value as needed) as the company identifier.
   d) Select `SMP101` for the class prefix.
   e) Select **Universal** as the device family product.
   f) Unselect **Use Core Data**.
4. Select a location in which to save the project and click **Create** to open it.

   Xcode creates a folder, `SMP101`, to contain the project file, `SMP101.xcodeproj`, and another `SMP101` folder, which contains a number of automatically generated files and a build folder.
5. Delete some of the automatically generated files created by default for the Xcode project.
   a) Delete the `SMP101` folder inside the top level of the SMP101 project:
      1. In Xcode, select the `SMP101` folder.
      2. Click **Remove References**.

3. In the Finder, manually delete the `SMP101` folder from the project folder.
4. Verify that only the `SMP101.xcodeproj` file and the build folder are in the `SMP101` folder.

6. Verify that the SDK and deployment targets are correct:
   a) Select SMP101 in Project Navigator and then select Build Settings.
   b) Under Project, select SMP101.
   c) Verify that Base SDK under Architectures is set to `Latest iOS (iOS 7.0)`.
   d) Scroll to the **Deployment** section and set the iOS Deployment Target to `iOS 4.3`.
   e) Select Targets > SMP101 and verify that those values are also set.
7. Copy the files from the `SMP101` folder on your Windows machine to the `SMP101` folder on your Mac that Xcode created to contain the SMP101 project:
   a) Connect to the Microsoft Windows machine where SAP Mobile Platform is installed
   b) From the Apple Finder menu, select **Go > Connect to Server**.
   c) Enter the name or IP address of the machine, for example, `smb://<machine DNS name>` or `smb://<IP Address>`, then click **Connect**.

      You see the shared directory.
   d) Copy the *SMP_HOME*`\MobileSDK23\ObjectAPI\iOS` folder from the SAP Mobile Platform installation directory to the `SMP101` folder on your Mac.
   e) On your Windows machine, navigate to the SMP101 mobile application project and copy the `Generated Code` folder to the SMP101 directory on your Mac.

Next
Add libraries, resources, and source code to the SMP101 Xcode project.

See also
- *Registering the Application Connection in SAP Control Center* on page 17

Adding Source Code Files, Libraries, and Resources to the Xcode Project

Once you set up the initial project in Xcode, add files from the SAP Mobile Platform folders you copied from your Windows machine.

1. In the Xcode Project Navigator, Ctrl-click the **SMP101 project**, then select **Add Files to "SMP101"**.

   Select the `Generated Code` folder, unselect **Copy items into destination group's folder (if needed)**, and click **Add**.

   The `Generated Code` folder is added to the project in the Project Navigator.
2. Ctrl-click the **Frameworks** group, then select **Add Files to "SMP101"**.

a) In the iOS folder you copied from the SAP Mobile Platform installation, navigate to the `Libraries/Debug-iphonesimulator` directory.
b) Select the `libAfariaSLL.a`, `libclientrt.a`, `libDatavault.a`, `libMO.a`, `libPerformanceLib.a`, `libsupClientUtil.a`, `libSUPObj.a`, `libSUPSupportability.a`, and `libsupUltralite.a` libraries.
c) Unselect **Copy items into destination group's folder (if needed)**.
d) Click **Add**.

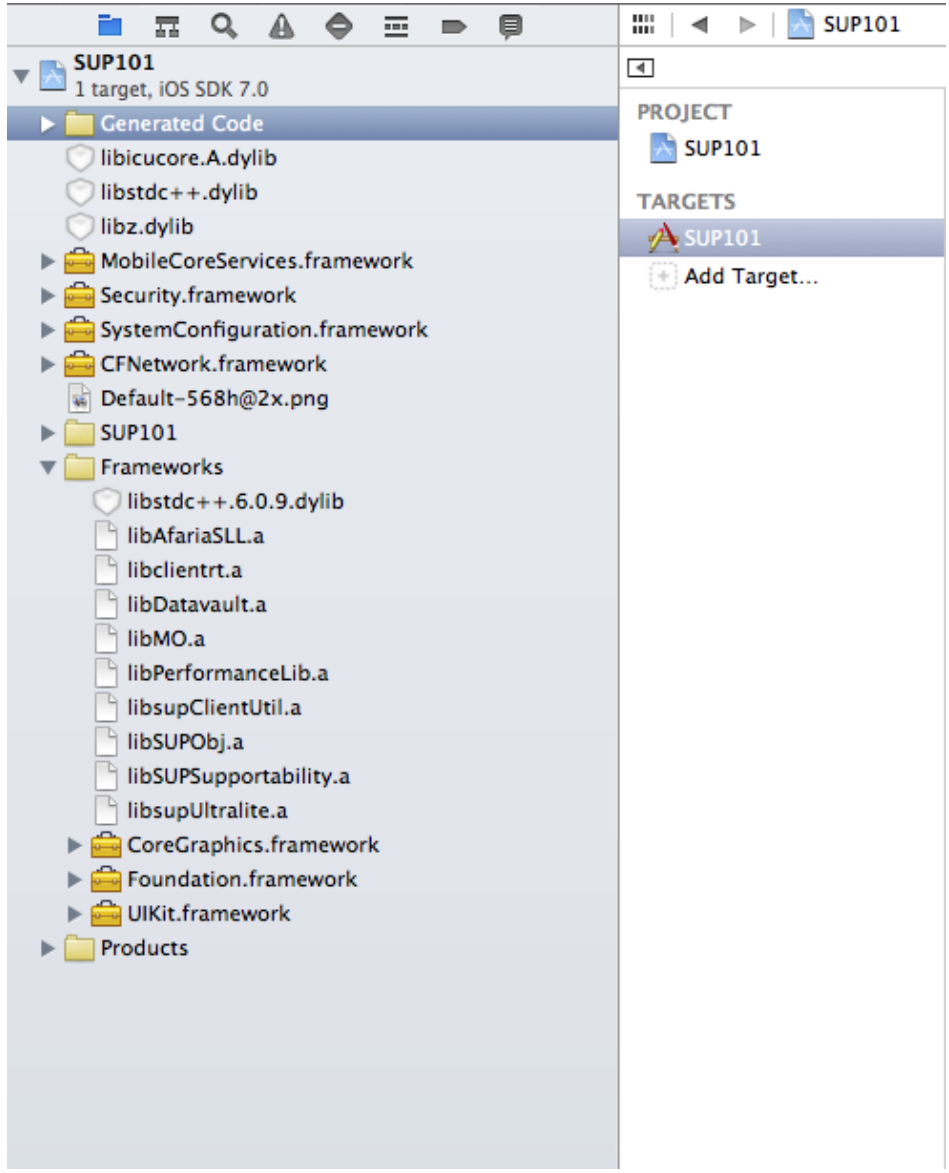
The libraries are added to the project in the Project Navigator.

Note: The library version corresponds to the configuration you are building. In this tutorial, you work with the libraries for the Debug version of the iPhone simulator.

3. Copy the `SMP101` folder from the `SMP101 iOS` Object API tutorial zip file to the `SMP101` project folder on your Mac.
4. Add the source code files that you copied from the SMP101 iOS Object API example project.
   a) In Xcode, Ctrl-click the SMP101 project and select **Add Files to "SMP101"**.

      Select the `SMP101` folder, unselect **Copy items into destination group's folder (if needed)**, and click **Add**.

   The project now looks like this:



Next
Configuring the build settings.

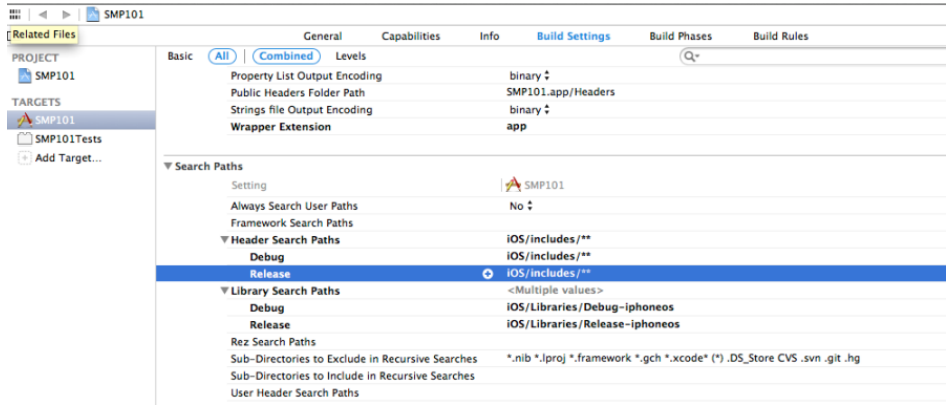
Configuring the Build Settings

Configure the build settings for the Xcode project, then build the project.

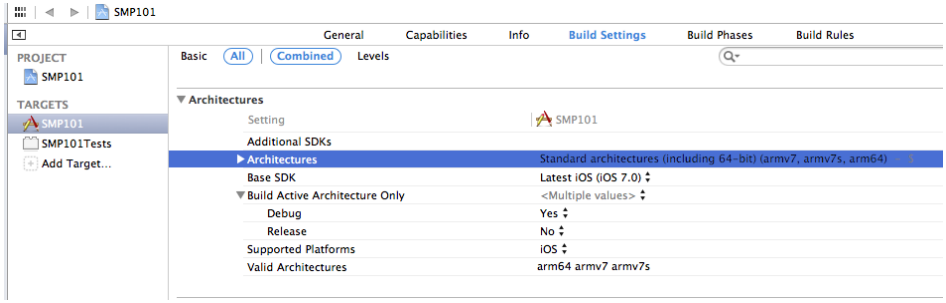
1. In the Project Navigator, under Project, select **SMP101** > **Build Settings** and scroll down to the Search Paths section. Enter the location of the iPhone simulator libraries in the Header Search Paths and Library Search Paths fields.

   `$SRCROOT` is a macro that expands to the directory where the Xcode project file resides. Adding this macro in front of the path is optional.

   - In Header Search Paths, enter the path to the `iOS/includes` directory, then select the recursive option. In this example, the path is indicated as `iOS/includes/**`.
   - In Library Search Paths, specify profiles for Debug and Release. In this example, the path is indicated as `"iOS/Libraries/$(CONFIGURATION)$(EFFECTIVE_PLATFORM_NAME)"`. Escape the path names using double quotes.

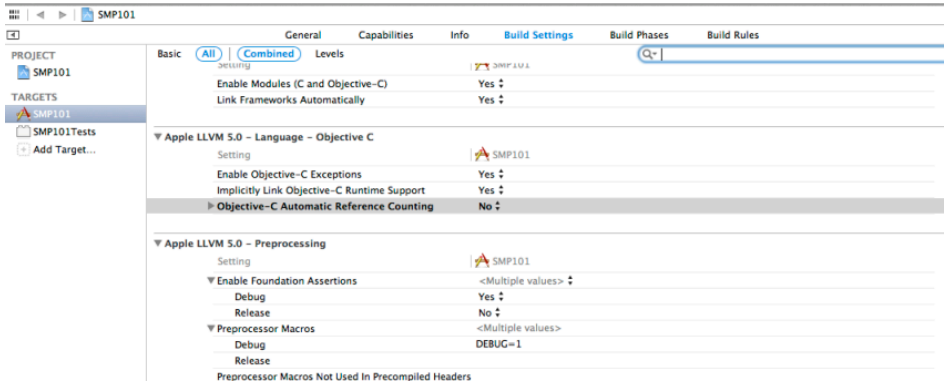


2. To build with 64-bit architectures, in the Project Navigator, under Target, select **SMP101** > **Build Settings**. Select **Architectures**, and scroll down to select **Standard architectures (including 64-bit)(armv7,armv7s,arm64)**.



3. In the Project Navigator, under Target, select **SMP101** > **Build Settings**, then expand the **Apple LLVM 5.0 Language - Objective C** section.

Set **Objective - C Automatic Reference Counting** to **No** to disable automatic reference counting.



4. In the Project Navigator, under Target, select **SMP101** > **Build Phases**, then expand the **Link Binary with Libraries** section.

   Click the **+** icon below the list, select the following libraries, and then click **Add** to add them to the project:

   - `CFNetwork.framework`
   - `CoreFoundation.framework`
   - `libicucore.A.dylib`
   - `libstdc++.6.0.9.dylib`
   - `libz.dylib`
   - `MobileCoreServices.framework`
   - `Security.framework`
   - `SystemConfiguration.framework`

5. In the Project Navigator, under Target, select **SMP101** > **Build Phases**, then expand the **Copy Bundle Resources** section. Select SMP101-info.plist and click on the **-** sign to remove it.
6. Select **Product > Clean** , then **Product > Build** to test initial project setup. If you correctly followed this procedure, you see a `Build Succeeded` message.

Registering the Application Connection in SAP Control Center

Register the iPhone simulator in SAP Control Center.

Prerequisites

Connect to SAP Control Center.

Task

1. Log in to SAP Control Center using the credentials you indicated during installation.
2. In SAP Control Center, select **View > Select > Mobile Server Cluster Management View**.
3. In the left pane, select **Applications**.
4. In the right pane, click **Application Connections**.
5. Click **Register**.
6. In the Register Application Connection window, enter the required information:
   - User name – `user1`
   - Template – `SMP101_admin`
   - Application ID – `SMP101`
   - Security configuration – `admin`
   - Logical role – leave blank or use the logical role when you assigned to this user during the deployment to Server
   - Domain – `default`
   - Activation code length – `3`
   - Activation expiration (hours) – `72`
   - Specify activation code – `123`
7. Click **OK**.

Next

In Xcode, view the application source files and walk through how they are created.

See also

- *Setting Up an iOS Client Application in Xcode* on page 12

Viewing the CallbackHandler and ApplicationCallbackHandler Files

Look at `CallBackHandler` and `ApplicationCallbackHandler` files in Xcode.

`CallbackHandler` is a subclass of `SUPDefaultCallbackHandler`, and listens for events sent from the server. It also implements the `SUPSyncStatusListener` protocol to get the synchronize status during the sync, and send notifications when synchronize is done (`ON_SYNC_STATE_DONE`) and/or if synchronize fails (`ON_SYNC_ERROR`). The header, `CallbackHandler.h`, is referenced in a number of classes in this application, so create it first. You can create new Objective-C class files from the main menu: **File > New > New File**.

`ApplicationCallbackHandler` is a subclass of `SUPApplicationDefaultCallback`, and it gets connection, registration, and device state change notifications.

There are two threads involved in the SMP101 application—the main thread, which is driven by the client application user interface controller, and the mobile object client access thread, which handles data synchronization with the server. In iOS, all code that updates the user interface must be called on the main thread, so SAP recommends, as a best practice, that you send notifications that might trigger changes to the interface from the main thread.

1. Click the `CallbackHandler.h` file to view the provided source code.
2. Click the `CallbackHandler.m` file to view the provided source code.
3. Click the `ApplicationCallbackHandler.h` file to view the provided source code.
4. Click the `ApplicationCallbackHandler.m` file to view the provided source code.

Creating the User Interface

Use Interface Builder to create and configure the user interface for the SMP101 application.

The SMP101 iOS Object API example project contains the source code for the user interface for the sample application. Although the user interface is built automatically when you add the source files to the Xcode project, you can walk through the rest of the tasks and view the source code to see how to use Interface Builder to build the sample application.

See also

- *Deploying the Device Application* on page 37

Viewing the SubscribeController View Controller

A view controller functions as the root view screen for the SMP101 mobile application.

The SubscribeController view controller automatically shows connection status and provides two buttons for action after the connection is made: one to synchronize data after connection to the device; and one to change the password. The **Synchronize** button will trigger data synchronization, and trigger the MenuListController view when synchronization is complete. The **Change Password** button will allow the user to change the password for connection to SAP Mobile Platform and trigger the PasswordPinViewController view.

In Xcode, you can create the view controller by creating a new file using the UIViewController subclass. Select **With XIB for user interface**. Xcode creates the corresponding `.h`, `.m`, and `.xib` files.

1. In the SMP101 Xcode project, click `SubscribeController.m` to view the logic for the view controller.

2. Click `SubscribeController.h` to view the header file.

See also

- *Creating the PasswordPinViewController* on page 31
- *Creating the MenuListController* on page 34
- *Creating the CustomerListController* on page 34
- *Adding the DetailController and Configuring the View* on page 35

Viewing the SMP101Appdelegate Files

The `SMP101Appdelegate.h` and `SMP101Appdelegate.m` files are created when you create the Xcode project; however, you deleted the automatically generated versions and replaced them with the ones added from the source code ZIP file.

The SMP101Appdelegate files make use of the `SUPApplication` and `SUPDataVault` APIs to show how to store and retrieve sensitive data (such as SAP Mobile Platform credentials) using a PIN.

The `applicationDidFinishLaunching` method checks to see if the application has been run before, then prompts the device user for a PIN to unlock the application; the SAP Mobile Platform user's password is also requested.

Control passes to the `initializeSMP101` method. This code sample does one of two things, depending on whether the application has been run before:

- If the application is running for the first time, the sample creates a new SUPDataVault secured with the user-provided PIN, to store the password and other items.
- If the application has been run before, the sample attempts to unlock the existing vault with the provided PIN. If this fails, the application displays an error dialog.

```
if(self.firstRun)
    {
        NSLog(@"Running the app for the first time.");



       // If the application is being run for the first time, we do
the following:
       //     1. Remove the messaging data vault created by earlier
versions of the application, if it exists.
        //      2. Remove the SMP101 data vault created by earlier
versions of the application, if it exists.
        //      3. Create the messaging vault using the PIN as the
password, leaving it unlocked for use by the messaging layer.
        //      4. Create the SMP101 data vault using the PIN as the
password, and store the SMP username/password credentials and a
database encryption key in the vault.
        @try
        {
            NSLog(@"Delete preexisting messaging vault");
            [SUPDataVault deleteVault:kMessagingDataVaultID];
```

```
        }
        @catch(NSException *e)
        {
            // Ignore any exception
        }
        @try {
            NSLog(@"Delete preexisting SMP101 data vault");
            [SUPDataVault deleteVault:kSMP101DataVaultID];
        }
        @catch(NSException *e)
        {
            // Ignore any exception
        }

        @try {
          NSLog(@"Create new SMP101 data vault and store credentials
and a generated encryption key");
           smp101vault = [SUPDataVault createVault:kSMP101DataVaultID
withPassword:self.pin withSalt:kSMP101DataVaultSalt]; // creates the
vault
            [smp101vault setString:@"password"
withValue:self.password];
            if (![smp101vault isLocked])
            {
                [smp101vault lock];
            }
        }
        @catch (NSException *exception) {
           NSLog(@"Exception in creating new SMP101 data vault: %@:
%@",[exception name], [exception reason]);
            [self showNoTransportAlert:kSMP101ErrorFailure];
            return;
        }
        @try {
            NSLog(@"Create new messaging vault and leave it
unlocked");
            messagingvault = [SUPDataVault
createVault:kMessagingDataVaultID withPassword:self.pin
withSalt:kDVStandardSalt];
        }
        @catch (NSException *exception) {
            NSLog(@"Exception in creating new messaging data vault:
%@: %@",[exception name], [exception reason]);
            [self showNoTransportAlert:kSMP101ErrorFailure];
            return;
        }

    }
    else
    {
       // If the application has been run before, we get the PIN from
the user, and use it to unlock the existing messaging data vault
        // (otherwise the messaging layer cannot start).
        NSLog(@"App has been run before.");
        @try {
            NSLog(@"Unlock messaging vault");
```

```
            messagingvault = [SUPDataVault
getVault:kMessagingDataVaultID];
            if ([messagingvault isLocked])
            {
                [messagingvault unlock:self.pin
withSalt:kDVStandardSalt];
            }
        }
        @catch (NSException *exception) {
           NSLog(@"Exception unlocking messaging data vault: %@: %@",
[exception name],[exception reason]);
            [self showNoTransportAlert:kSMP101ErrorBadPin];
            return;
        }

    }
```

This code sample sets up the Application API settings for connection to the SAP Mobile Server and registers with the SAP Mobile Server.

```
// Add the observer to listen for ON_REGISTER_SUCCESS or
ON_CONNECT_SUCCESS for the first run or
// subsequent runs respectively. Also for ON_CONNECT_FAILURE, and
ON_REGISTER_FAILURE. Refer to the comments in
registerObserverForCallbackNotifications.
// The observer must be added before the call to registerApplication,
but after applicationIdentifier is
// set and the messaging vault unlocked. (AppIdentifier being set and
the vault being unlocked are prerequisites to
// calling if ([SUPApplication registrationStatus] ==
SUPRegistrationStatus_REGISTERED) which is used in
registerObserverForCallbackNotifications.

    [self.viewController registerObserverForCallbackNotifications];


    @try {
        smp101vault = [SUPDataVault getVault:kSMP101DataVaultID];
        if ([smp101vault isLocked])
        {
            [smp101vault unlock:self.pin
withSalt:kSMP101DataVaultSalt];
        }

       // Register callback handlers. This should be done before any
other SMP code is called.
        [SMP101SMP101DB registerCallbackHandler:[CallbackHandler
getInstance]];
        [app setApplicationCallback:[ApplicationCallbackHandler
getInstance]];


        // Setup the connection properties and login credentials
required for registration.
        SUPConnectionProperties* props = app.connectionProperties;
        [props setServerName:self.serverName];
```

```
        [props setPortNumber:[self.serverPort intValue]];
        [props setUrlSuffix:@""];
        [props setFarmId:self.farmID];

        SUPLoginCredentials* login = [SUPLoginCredentials
getInstance];
        if(self.manualRegistration)
        {
            login.username = self.connectionName;
            login.password = nil;
            props.activationCode = self.activationCode;
        }
        else
        {
            login.username = self.userName;
            login.password = [smp101vault getString:@"password"];
            props.activationCode = nil;
        }
        props.loginCredentials = login;
```

This code sample does one of two things depending on whether the application has been run before:

- If the application is running for the first time, the sample creates the SMP101 database, generates an encryption key, and stores it in the data vault.
- If the application has been run before, the sample retrieves the encryption key from the data vault, and sets it in the connection profile so the database can be reused.

```
// Get the connection profile for the database.
        SUPConnectionProfile *cp = [SMP101SMP101DB
getConnectionProfile];
        [cp enableTrace:NO];

        // Delete any existing database from previous versions.
        if(self.firstRun && [SMP101SMP101DB databaseExists])
        {
            [SMP101SMP101DB deleteDatabase];
        }


       // Create the database if required and set the encryption key.
        if(![SMP101SMP101DB databaseExists])
        {
            [SMP101SMP101DB createDatabase];
           // We need to generate a new encryption key to encrypt the
DB
            [SMP101SMP101DB generateEncryptionKey];
            // Store the encryption key in the data vault for future
use.
            SUPConnectionProfile *cp = [SMP101SMP101DB
getConnectionProfile];
            [smp101vault setString:@"encryptionkey" withValue:[cp
getEncryptionKey]];
        }
        else
        {
```

```
          // When we are create the database from scratch, we set the
database encryption key in generateEncryptionKey.
          // If we were using the database from a previous run of the
app and not creating it each time, an application should
            // run the code below instead to successfully access a
previously encrypted database by retrieving the encryption key
            // from the datavault and setting it in the connection
profile.
           NSString *key = [smp101vault getString:@"encryptionkey"];
            NSLog(@"Got the encryption key: %@",key);
            [cp setEncryptionKey:key];
        }

       // Set the synchronization configuration required to sync with
the server.
        SUPConnectionProfile *sp = [SMP101SMP101DB
getSynchronizationProfile];
        [sp setDomainName:@"default"];
        [sp enableTrace:YES];
       // by default the AsyncReplay is enabled. We will turn it off.
This will make the next syncrhonization a blocking call.
        [sp setAsyncReplay:NO];
        [sp setUser:self.userName];
        [sp setPassword:[smp101vault getString:@"password"]];


    }
    @catch (SUPPersistenceException * pe) {
        NSLog(@"%@: %@", [pe name],[pe message]);
        [self showNoTransportAlert:kSMP101ErrorFailure];
        return;
    }
    @catch (NSException* e) {
        NSLog(@"%@: %@", [e name],[e reason]);
        [self showNoTransportAlert:kSMP101ErrorFailure];
        return;
    }
    @finally
    {
        if (![smp101vault isLocked])
        {
            [smp101vault lock];
        }
    }

    @try {
        // Initialize generated package database class with this
application instance.
        [SMP101SMP101DB setApplication:app];

        if ([SUPApplication registrationStatus] !=
SUPRegistrationStatus_REGISTERED)
        {
        // Register the application with the server.
        [app registerApplication:300];
        }
```

```
        else
        {
        // already registered, start connection.
        [app startConnection:300];
        }

        // Update the value of self.firstRun, We have created the
vault and registered with server at this point.
        self.firstRun = (![MessagingClientLib isMessagingDBExist] ||
                         ![SUPDataVault
vaultExists:kSMP101DataVaultID]);

    }
    @catch (SUPApplicationTimeoutException* tex)
    {
        NSLog(@"%@: %@", [tex name],[tex message]);
        [self showNoTransportAlert:kSMP101ErrorFailure];
        return;
    }
    @catch (NSException *e)
    {
       // When we are faced with a registration error or connection
error, the 'onRegistrationStatusChanged'
      // or on 'onConnectionStatusChanged' callbacks are triggered in
which we send the ON_CONNECT_FAILURE
       // notification or the ON_REGISTER_FAILURE notification to
handle it
      //  and show the alert window to the user.So we don't have to do
it again here.
       // For all other failures, other than the timeout exception
above , we will handle it here.
        if ([SUPApplication registrationStatus] ==
SUPRegistrationStatus_REGISTRATION_ERROR)
        {
            return;
        }
        if ([SUPApplication connectionStatus] ==
SUPConnectionStatus_CONNECTION_ERROR)
        {
            return;
        }

        NSLog(@"%@: %@", [e name],[e reason]);
        [self showNoTransportAlert:kSMP101ErrorFailure];
        return;
    }
```

When you run the application for the first time, the ON_REGISTER_SUCCESS notification is registered, and for subsequent runs, the ON_CONNECT_SUCCESS notification is registered. In the above code snippet, the call to the `registerObserverForCallbackNotifications` method (in SubscribeController) registers the appropriate observer depending on whether or not you are running the application for the first time.

If the application is running for the first time, `registerStatus` is not SUPRegistrationStatus_REGISTERED, so `[app registerApplication:300]` is called to start a connection with the server and register the application. If the registration is successful, the `onRegistrationStatusChanged:(SUPRegistrationStatusType)registrationStatus:(int32_t)errorCode :(NSString*)errorMessage` callback is called, resulting in an ON_REGISTER_SUCCESS notification being posted. Upon receiving this notification, the observer for the ON_REGISTER_SUCCESS notification calls `onConnectSuccess:(NSNotification *)notification` in the SubscribeController. This enables the **Synchronize** button in the application, allowing you to initiate a sync with the server.

For subsequent runs, since the application is already registered, `registerStatus` is SUPRegistrationStatus_REGISTERED, and the call to `[app registerApplication:300]` just starts a connection with the server. In this case, when the connection is successful, the`onConnectionStatusChanged:(SUPConnectionStatusType)connectionStatus:(int32_t)errorCode :(NSString*)errorMessage` callback is called, resulting in an ON_CONNECT_SUCCESS notification being posted. Upon receiving this notification, the observer for the ON_CONNECT_SUCCESS notification calls `onConnectSuccess:(NSNotification *)notification` in the SubscribeController. This enables the **Synchronize** button in the application, allowing you to initiate a sync with the server.

The process above ensures that you initiate a sync with the server only after a registration (and a connection) is made for the first run. For all other runs you need to initiate the sync only after you make a connection.

If you are connecting to the SAP Mobile Server through a Relay Server, then you must provide additional information for the database synchronization profile:

- Add the certificate file provided by the Relay Server to the `Resource` folder of your Xcode project.
- Add this code:

```
SUPConnectionProfile *sp = [SMP101SMP101DB
getSynchronizationProfile];
[sp setNetworkProtocol:@"https"]; // or http
[sp setPortNumber:443]; // if http then corresponding port
[sp
setNetworkStreamParams:@"trusted_certificates=certificateName;com
pression=zlib;url_suffix=urlsuffixProvidedByTheRelayServer"];
```

  - **NetworkProtocol –** http or https.
  - **PortNumber –** the correct port number for the selected NetworkProtocol.
  - **NetworkStreamParams –** certificateName: the name of the certificate you added in the `Resource` folder.

    urlsuffixProvidedByTheRelayServer: the URL suffix provided by the Relay Server

Configuring the SubscribeController View

Use Interface Builder to configure the `SubscribeController.xib` file and create the user interface. Although the provided XIB file is already configured, you can walk through the steps to see how to create the interface.

1. Click the `SubscribeController.xib` file to reveal a view of the (presently empty) screen in the right pane and the following three items represented by icons in the middle pane:
   - File's Owner – the object that is set to be the owner of the user interface, which is typically the object that loads the interface. In this tutorial, this is the SubscribeController.

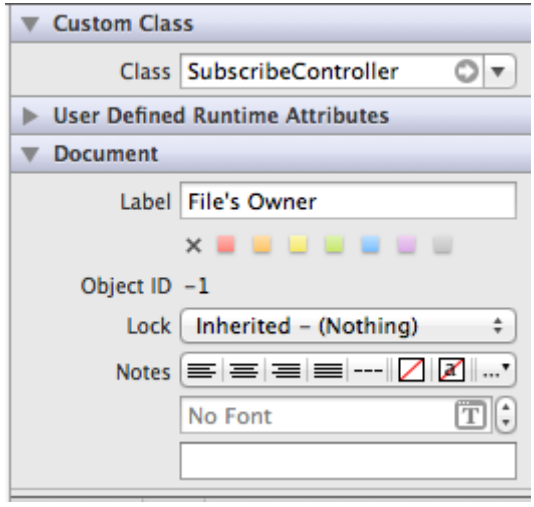
   - First Responder – the first responder proxy object handles events. Connecting an action to the first responder means that when the action is invoked, it is dynamically sent to the responder chain.

   - View – appears in a separate window to allow editing.

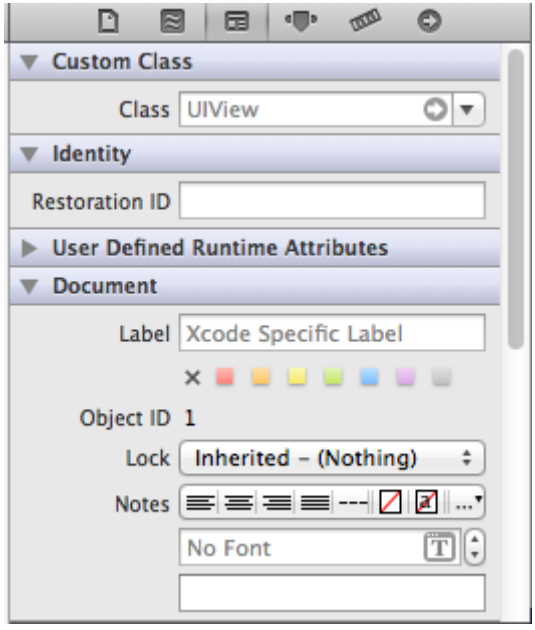
     

2. Select the **File's Owner** icon, click **View** in the utility area, click **Show the Identity Inspector**, and make sure `SubscribeController` appears in the Class field under Custom Class.

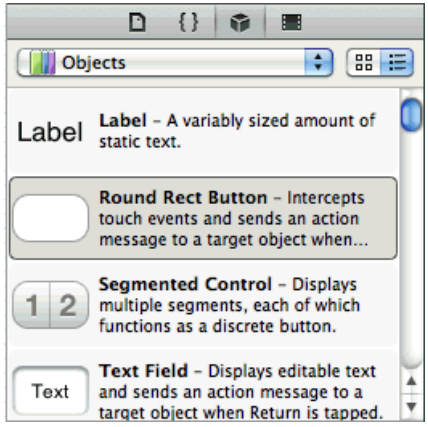
   

This tells Interface Builder the class of the object to allow you to make connections to and from the File's Owner.

3. Click the **View** icon, and in the Identity Inspector panel, and make sure `UIView` appears in the Class field under Custom Class.



4. To create a **Subscribe** button, select **View > Utilities > Show Object Library**.
   a) In the Object Library pane, select the **Round Rect Button** item, and drag it onto the view.

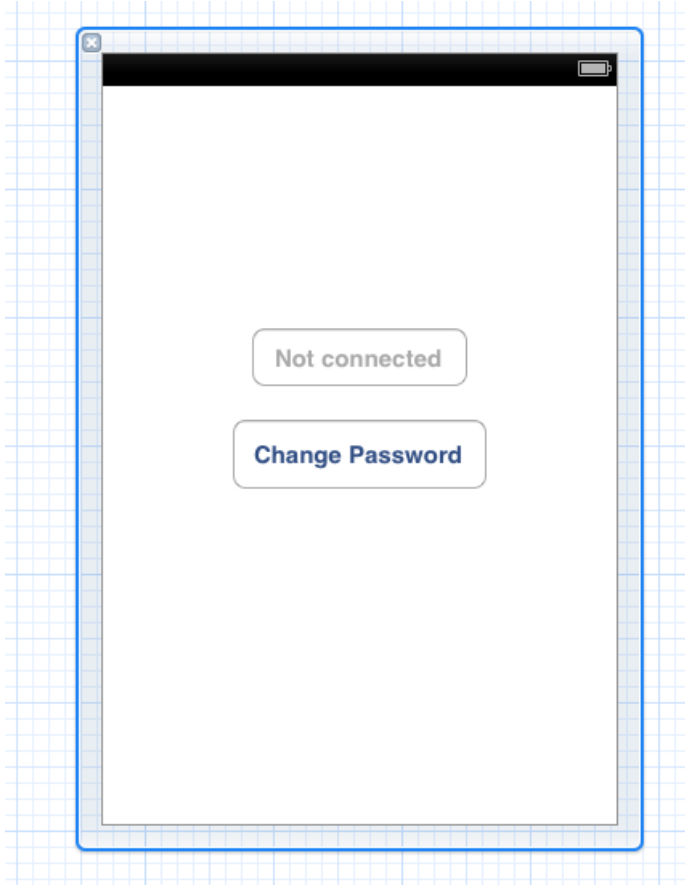


   b) Double-click it, enter `Not Connected`, and press Enter.
5. Repeat Step 4 to create a **Change Password** button.

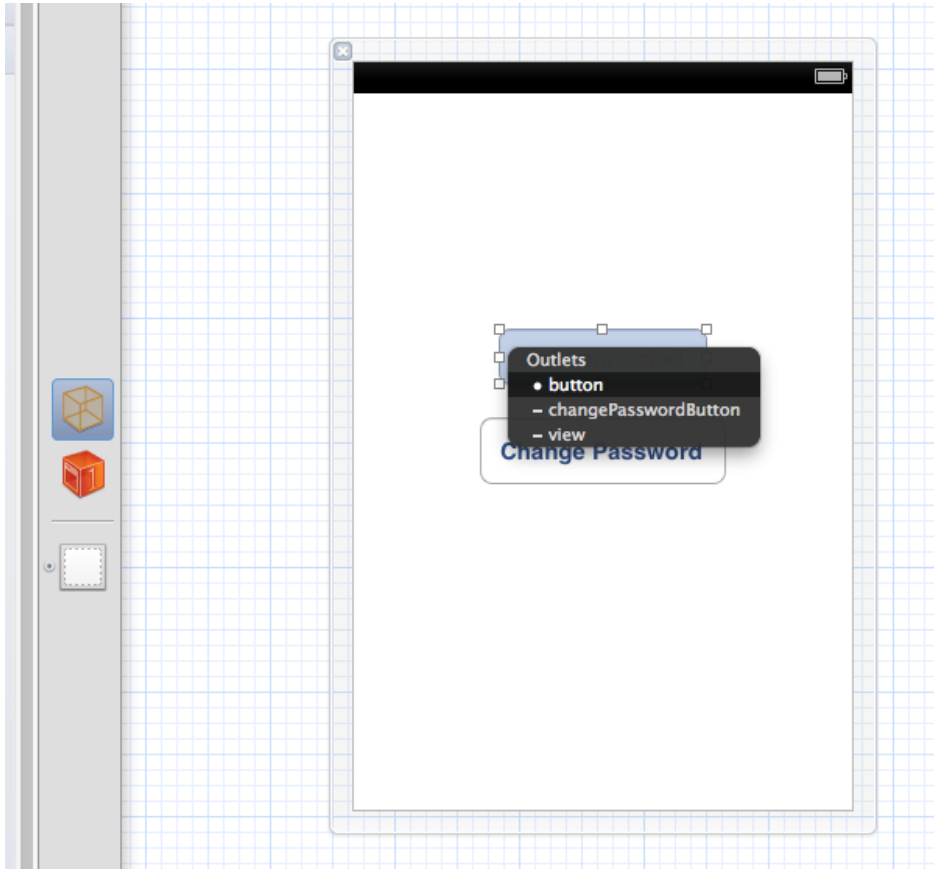
6. To make connections to the user interface from the view controller, the SubscribeController.h file contains the outlets, property declarations for the instance variables, and a declaration for the action method.

```
-(IBAction)buttonPressed:(id)sender;
-(IBAction)changePasswordButtonPressed:(id)sender;

@property (nonatomic, retain) IBOutlet UIButton *button;
@property (nonatomic, retain) IBOutlet UIButton
*changePasswordButton;
@property (nonatomic, retain) MenuListController *menuController;
```

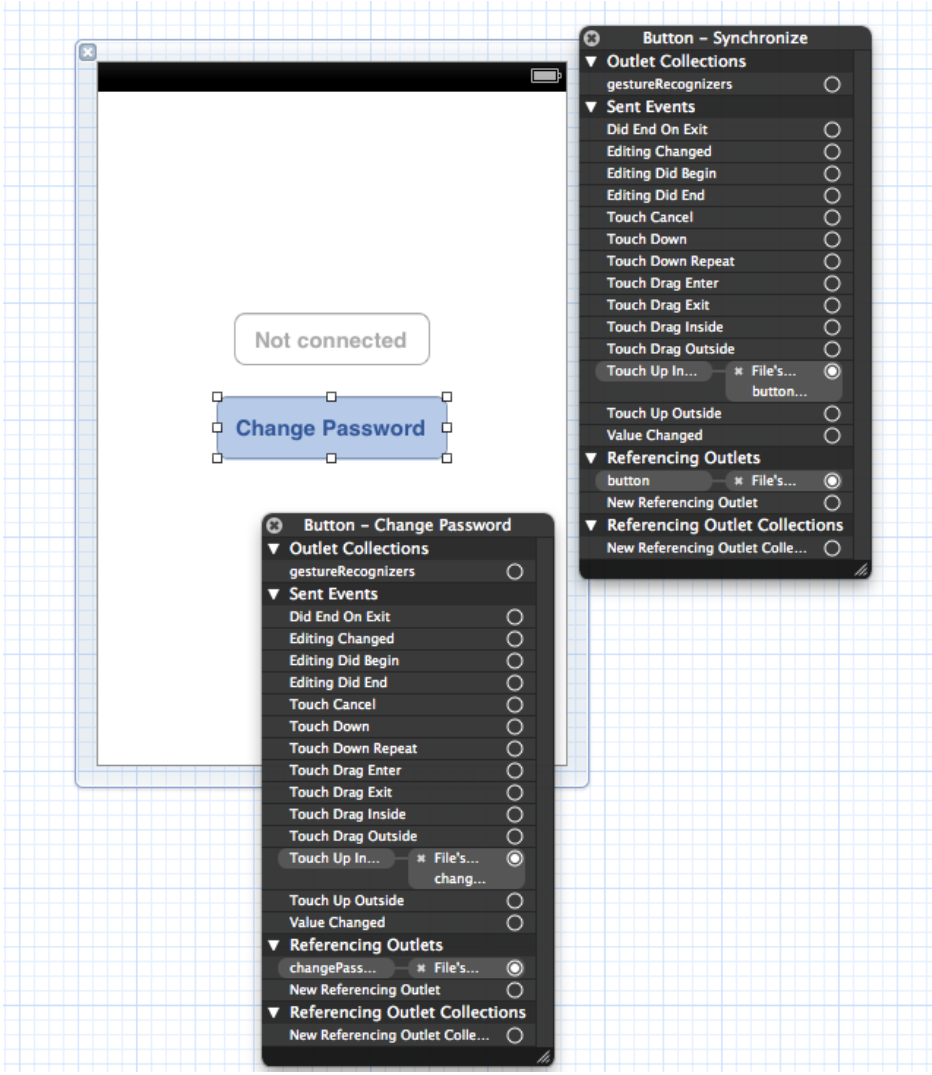
   Save any changes to `SubscribeController.h` and `SubscribeController.m`.



7. Ctrl-drag on the **File's Owner** icon in the middle pane to the **Subscribe** button and select outlet button. Repeat for the **changePassword** button.



8. Ctrl-click the **Subscribe** button to show the inspector.
9. Drag from the circle to the right of **Touch Up Inside** to the **File's Owner** icon and release, then click **buttonPressed** to establish a connection between the **Subscribe** button and the button's action method. Repeat for the **changePassword** button, and select **changePasswordButtonPressed**, to establish a connection between the **changePassword** button and the button's action method:



Creating the PasswordPinViewController

Create the password pin view.

The PasswordPinViewController allows mobile device users to input the PIN to lock and unlock the datavault and the SAP Mobile Platform password required for data synchronization, and/or for automatic registration when an application is first launched. The password is saved in the datavault and only the PIN is required to unlock the datavault for subsequent launches.

The source files you added from the SMP101 iOS Object API example project contain the `PasswordPinViewController.h`, `PasswordPinViewController.m`, and `PasswordPinViewController.xib` files that create the password pin view. This file also supports creating a new pin. To create these files manually in Xcode, you would create a new file using the UIViewController subclass template, then indicate it is a subclass of `UIViewController`. Select **With XIB for user interface**.

Although the provided XIB file is already configured, you can walk through the steps to see how to create the interface.

1. Click the `PasswordPinViewController.xib` file to open Interface Builder.
2. Select **View** > > **Utilities** > **Show Object Library**.
3. In the Object Library pane, select the **Text Field** item, and drag it onto the view two times to create two text fields aligned vertically to the right of the screen.

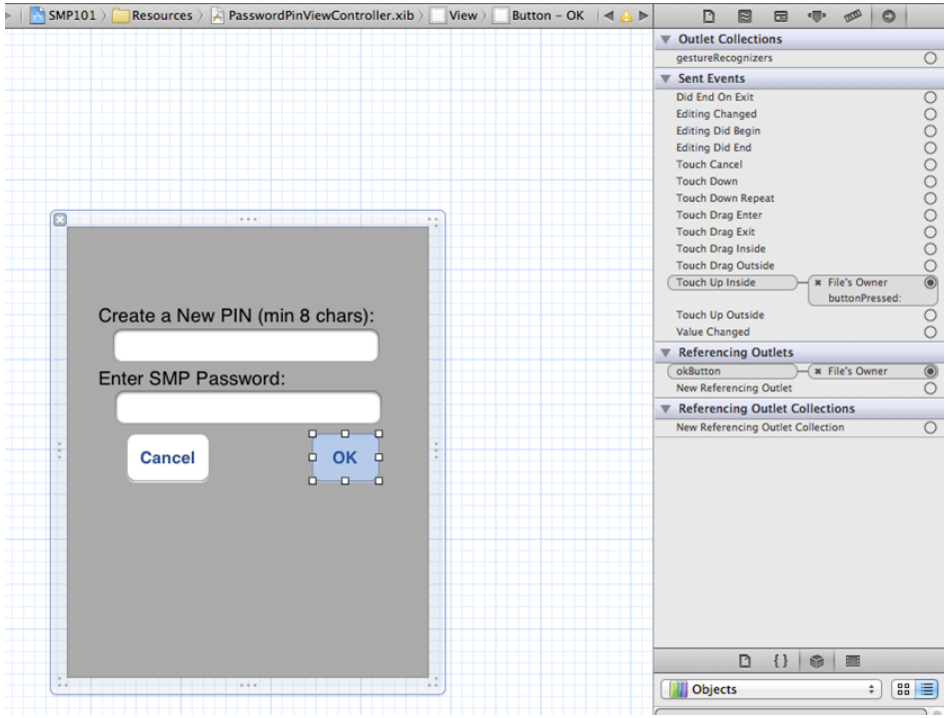
   You can resize the text fields using the resize handles, and position the button by dragging it to the desired location.
4. In the Object Library panel, select the **Label** item, and drag it onto the view two times to create two labels to the above and aligned with the two text fields. Replace the default `Label` text with:
   - Create a New PIN (min 8 chars)
   - Enter SMP Password
5. In the Object Library panel, select the **Round Rect Button** item, drag it onto the view, and rename it `Cancel`. Add an `OK` button in the same way.

   To make connections to the user interface from the view controller, the `PasswordPinViewController.h` file contains the outlets, property declarations for the instance variables, and a declaration for the action method.

```
@interface PasswordPinViewController : UIViewController
@property (nonatomic, retain) IBOutlet UILabel *pinLabel;
@property (nonatomic, retain) IBOutlet UITextField
*datavault_pin;
@property (nonatomic, retain) IBOutlet UILabel *passwordLabel;
@property (nonatomic, retain) IBOutlet UITextField *smp_password;

@property (nonatomic, retain) IBOutlet UIButton *cancelButton;
@property (nonatomic, retain) IBOutlet UIButton *okButton;

@property (nonatomic, assign) UIViewController *mvc;

-(IBAction)buttonPressed:(id)sender;
-(IBAction) backgroundClick: (id) sender;
@end
```

   Save any changes that you make to the `PasswordPinViewController.h` and `PasswordPinViewController.m` files.

6. Click the `DetailController.xib` file to open it in Interface Builder, click the text field you created for the pin, and select **View > Utilities > Show Attributes Inspector**.
7. Ctrl-drag from the **File's Owner** icon in the middle pane to each of the text fields and select the **datavault_pin** and **smp_password** outlets, respectively, to create connections between the text fields and the outlets defined in the `PasswordPinViewController.m` file.
8. Select **View > Utilities > Show Connections Inspector** to confirm that the outlets have been correctly configured:



9. Ctrl-drag from the **File's Owner** icon in the middle pane to the **Cancel** button and select **buttonPressed** to connect the **Cancel** button with the `Touch Up Inside` event.
10. Repeat step 9 for the **OK** button.

See also

- *Viewing the SubscribeController View Controller* on page 19
- *Creating the MenuListController* on page 34
- *Creating the CustomerListController* on page 34
- *Adding the DetailController and Configuring the View* on page 35

Creating the MenuListController

Create the menu list view.

The source files you added from the SMP101 iOS Object API example project contain the `MenuListController.h`, `MenuListController.m`, and `MenuListController.xib` files that create the menu list view. To create these files manually in Xcode, create a new file using the UIViewController subclass template, then indicate it is a subclass of `UITableViewController`. Select **With XIB for user interface**.

1. View the `MenuListController.h` file.
2. View the `MenuListController.m` file.

   `MenuListController.m` is a table view controller that displays two menu items: List and Create. Tap a row to move to the corresponding screen.

See also

- *Viewing the SubscribeController View Controller* on page 19
- *Creating the PasswordPinViewController* on page 31
- *Creating the CustomerListController* on page 34
- *Adding the DetailController and Configuring the View* on page 35

Creating the CustomerListController

Create the customer list view.

The source files you added from the SMP101 iOS Object API example project contain the `CustomerListController.h`, `CustomerListController.m`, and `CustomerListController.xib` files which create the customer list view. To create these files manually in Xcode, create a new file using the UIViewController subclass template, then indicate it is a subclass of `UITableViewController`. Select **With XIB for user interface**.

1. View the `CustomerListController.h` file.
2. View the `CustomerListController.m` file.

   `CustomerListController.m` is a table view controller that displays the customer data in the client database. The `viewWillAppear` method uses the Object API to query the database for a list of all Customer objects, and builds an NSArray that is used by this class as the datasource for displaying the table view.

   If a row is tapped, the `accessoryButtonTappedForRowWithIndexPath` method is executed, which pushes a `DetailController` onto the stack to display additional information and allow the data to be modified.

See also

- *Viewing the SubscribeController View Controller* on page 19
- *Creating the PasswordPinViewController* on page 31
- *Creating the MenuListController* on page 34
- *Adding the DetailController and Configuring the View* on page 35

Adding the DetailController and Configuring the View

Create the `DetailController.xib`.

The detail controller view displays information about a single customer in the client database. The source files you added from the SMP101 iOS Object API example project contain the `DetailController.h`, `DetailController.m`, and `DetailController.xib` files that create the customer detail view. This file also supports creating a new customer or deleting an existing customer. To create these files manually in Xcode, you would create a new file using the UIViewController subclass template, then indicate it is a subclass of `UIViewController`. Select **With XIB for user interface**.

Although the provided XIB file is already configured, you can walk through the steps to see how to create the interface.

1. Click the `DetailController.xib` file to open Interface Builder.
2. Select **View > Utilities > Show Object Library**.
3. In the Object Library pane, select the **Text Field** item, and drag it onto the view three times to create three text fields aligned vertically to the right of the screen.

   You can resize the text fields using the resize handles, and position the button by dragging it to the desired location.
4. In the Object Library panel, select the **Label** item, and drag it onto the view three times to create three labels to the left of and aligned with the three text fields. Replace the default `Label` text with:
   - First Name
   - Last Name
   - Phone
5. In the Object Library panel, select the **Round Rect Button** item, drag it onto the view, and rename it `Submit`. Add a `Delete` button in the same way.

   To make connections to the user interface from the view controller, the `DetailController.h` file contains the outlets, property declarations for the instance variables, and a declaration for the action method.

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

#import "SMP101Customer.h"
#import "CallbackHandler.h"

@class CallbackHandler;
```

```
@interface DetailController : UIViewController {
    BOOL _deleteRecord;
}

@property (nonatomic, retain) IBOutlet UITextField *fname;
@property (nonatomic, retain) IBOutlet UITextField *lname;
@property (nonatomic, retain) IBOutlet UITextField *phone;
@property (nonatomic, retain) SMP101Customer *originalObj;
@property (nonatomic, retain) IBOutlet UIButton *submitButton;
@property (nonatomic, retain) IBOutlet UIButton *deleteButton;
@property (nonatomic, retain) IBOutlet UILabel *label;
@property (nonatomic, assign) BOOL deleteRecord;


-(IBAction)buttonPressed:(id)sender;
-(IBAction) keyBoardOff : (id) sender;
-(void)keyBoardOff;
-(void)cleanForm;

- (void)setupNotification;
- (void) reeplaySuccess:(NSNotification *)notification;
- (void) reeplayFailure:(NSNotification *)notification;

@end
```

6. View the `DetailController.m` file.

   This class displays detailed information about a single customer in the client database. The information can be edited. If the data is changed and the Submit button is pressed, the `buttonPressed` method uses Object API calls to save the changes in the client database, send the changes to the server, and disable the Submit button.

   If the server accepts the changes, the callback handler posts an ON_REPLAY_SUCCESS notification, which causes the `onReplaySuccess` notification handler to run. The cached UI data is refreshed from the database and the Submit button is reenabled.

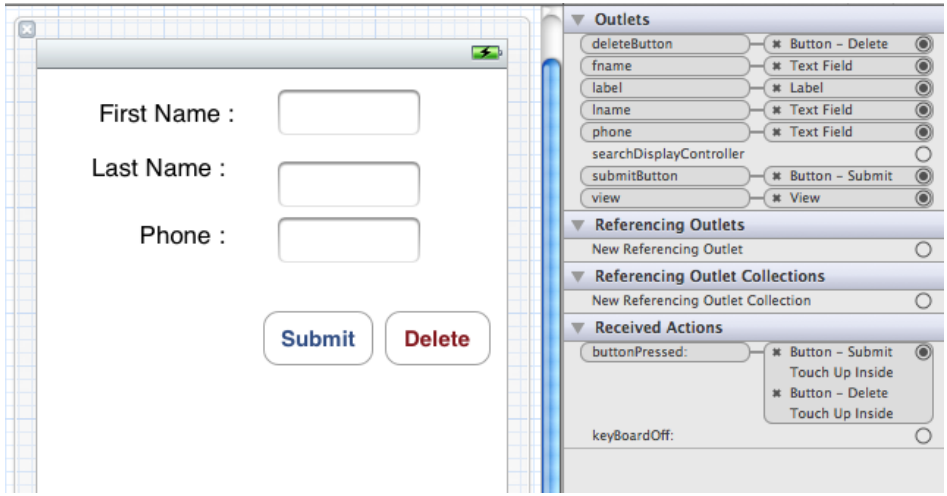
   This class also registers for the ON_REPLAY_FAILURE notification to handle the case where the server rejects the changes, or an error occurs on the server side.

   If you press the **Delete** button, the `buttonPressed` method uses the Object API calls to delete the record, then initiates a synchronization call to send the delete request to the server. If the server accepts the changes, the callback handler posts an ON_REPLAY_SUCCESS notification and the list page is shown.

   If the Create option is selected from the menu list, the DetailController is loaded with an empty form. The Submit button is called Create. If you fill out the form and press the **Create** button, a new record is created in the local database and a synchronization call is initiated. If the server accepts the new record, an ON_REPLAY_SUCCESS notification is posted.

7. Click the `DetailController.xib` file to open it in Interface Builder, click the **First Name** text field, and select **View > Utilities > Show Attributes Inspector**.

8. In the Attributes Inspector pane, scroll to the View section and enter `1` in the Tag field.
9. Set the tags for the Last Name and Phone text fields to `2` and `3` respectively.
10. Ctrl-drag from the **File's Owner** icon in the middle pane to each of the text fields and select the **fname**, **lname**, and **phone** outlets, respectively, to create connections between the text fields and the outlets defined in the `DetailController.m` file.
11. Select **View > Utilities > Show Connections Inspector** to confirm that the outlets have been correctly configured:



12. Ctrl-drag from the **File's Owner** icon in the middle pane to the **Submit** button and select **submitButton**.
13. Repeat steps 6 – 12 for the Delete button as for the Submit button.

See also

- *Viewing the SubscribeController View Controller* on page 19
- *Creating the PasswordPinViewController* on page 31
- *Creating the MenuListController* on page 34
- *Creating the CustomerListController* on page 34

Deploying the Device Application

Deploy the SMP101 application to the iPhone simulator for testing.

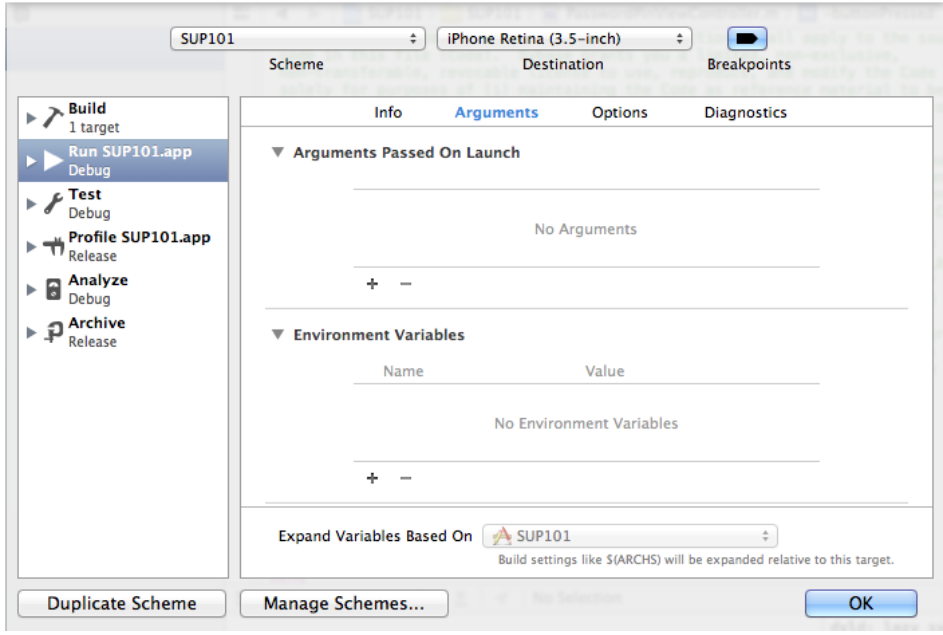
Prerequisites

Register an application connection in SAP Control Center.

You must be connected to the server where the mobile application project is deployed.

Task

1. From the top menu, select **Product > Scheme > Edit Scheme** and change to **iOS 7.0 Simulator**.



2. Select **Product > Build** then **Product > Run** to build the project and start the iPhone simulator.
3. In the iPhone applications screen, open the **SMP101** application.
   When you run the application for the first time, it exits immediately with a dialog asking you to enter the application settings in the Settings application.
4. In the iPhone simulator, go to **Settings > SMP101** to enter the connection settings.
   - **SMP Server** – the machine that hosts the server where the SMP101 mobile application project is deployed.
   - **SMP Server Port** – SAP Mobile Server port number. The default is 5001.
   - **Farm ID** – an identifier for the SAP Mobile Server Cluster for which a Relay Server manages requests. For this tutorial, accept the default value `0`.
   - **SMP Username** – the user to be authenticated, `supAdmin`.

If the Manual registration switch is off, the application attempts an automatic registration, creating an application registration with the same name as the SAP Mobile Platform user name ("supAdmin" in this example). This allows a client with a valid SAP Mobile Platform user name and password to connect and register with the server without manual registration.

If the Manual registration switch is on, the connection name and activation code must be filled in, and must match an application connection that has already been created in SAP Control Center (see *Registering the Application Connection in SAP Control Center*).

5. In the iPhone applications screen, reopen the **SMP101** application.
   Enter a pin with which to securely store your SAP Mobile Platform password, and a database encryption key that is generated when the application launches. For subsequent launches of the application, you need only enter the PIN.

6. Enter a PIN, and enter the password for the SAP Mobile Platform user name entered in step 4.
7. Click **Synchronize**.
8. Click **List**.
9. Select a customer record from the customer list and double-click to open the detail view. The customer detail shows the First Name, Last Name, and Phone.
10. Change the First Name to something else, and click **Update**.

See also

- *Creating the User Interface* on page 19

Learn More About SAP Mobile Platform

Once you have finished, try some of the other samples or tutorials, or refer to other development documents in the SAP Mobile Platform documentation set.

Check the Product Documentation Web site regularly for updates: *http://sybooks.sybase.com/sybooks/sybooks.xhtml?id=1289&c=firsttab&a=0&p=categories*, then navigate to the most current version.

Tutorials

Try out some of the other getting started tutorials available on the Product Documentation Web site to get a broad view of the development tools available to you.

Example Projects

An example project contains source code for its associated tutorial. It does not contain the completed tutorial project. Download example projects from the SAP® Community Network (SCN) at *http://scn.sap.com/docs/DOC-8803*.

Samples

Sample applications are fully developed, working applications that demonstrate the features and capabilities of SAP Mobile Platform.

Check the SAP® Development Network (SDN) Web site regularly for new and updated samples: *https://cw.sdn.sap.com/cw/groups/sup-apps*.

Online Help

See the online help that is installed with the product, or available from the Product Documentation Web site.

Developer Guides

Learn best practices for architecting and building device applications:

- *Mobile Data Models: Using Data Orchestration Engine* – provides information about using SAP Mobile Platform features to create DOE-based applications.
- *Mobile Data Models: Using Mobile Business Objects* – provides information about developing mobile business objects (MBOs) to fully maximize their potential.
- *SAP Mobile WorkSpace: Mobile Business Object Development* – provides information about using SAP Mobile Platform to develop MBOs and generate Object API code that can be used to create native device applications and Hybrid Apps.

Use the appropriate API to create device applications:

- *Developer Guide: Android Object API Applications*
- *Developer Guide: BlackBerry Object API Applications*

- *Developer Guide: iOS Object API Applications*
- *Developer Guide: Windows and Windows Mobile Object API Applications*
- *Developer Guide: Hybrid Apps*
- *Developer Guide: OData SDK*
- *Developer Guide: REST API Applications*

Customize and automate:

- *Developer Guide: SAP Mobile Server Runtime > Management API* – customize and automate system administration features.

Javadoc and HeaderDoc are also available in the installation directory.

Index