SYBASE®
An SAP® Company

Tutorial: iOS Object API Application Development

Sybase Unwired Platform 2.1 ESD #3

DOCUMENT ID: DC01213-01-0213-04
LAST REVISED: July 2012

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

Contents

Sybase Unwired Platform Tutorials

The Sybase® Unwired Platform tutorials demonstrate how to develop, deploy, and test mobile business objects, device applications, and mobile workflow packages. You can also use the tutorials to demonstrate system functionality and train users.

Tip: If you want to see the final outcome of a tutorial without performing the steps, the associated example project is available on SAP® Community Network: *http://scn.sap.com/docs/DOC-8803*. To perform any other tutorial for which this tutorial is a prerequisite, you must deploy the MBOs developed in this tutorial.

- 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*
- 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 Mobile Object API Application Development*
- Create a mobile business object, then develop a mobile workflow package that uses it:
  - *Tutorial: Mobile Workflow Package Development*

Getting Started with Unwired Platform

Install and learn about Sybase Unwired Platform and its associated components.

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

1. *Installing Sybase Unwired Platform*

   Install Sybase Mobile SDK and Sybase Unwired Platform Runtime.
2. *Starting Sybase Unwired Platform Services*

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

   Start the development environment, where you can create mobile business objects (MBOs), manage EIS data sources and Unwired Server connections, develop Mobile Workflow applications, and generate Object API code.
4. *Connecting to Sybase Control Center*

   Open the Sybase Control Center Administration Console to manage Unwired Server and its components.
5. *Learning Unwired WorkSpace Basics*

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

Installing Sybase Unwired Platform

Install Sybase Mobile SDK and Sybase Unwired Platform Runtime.

Before starting this tutorial, install all the requisite Unwired Platform components. See the Sybase Unwired Platform documentation at *http://sybooks.sybase.com*:

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

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

- Development support for native Object API applications, HTML5/JS Hybrid (Mobile Workflow) applications, and native OData SDK applications.
- Sybase Unwired WorkSpace, the Eclipse-based development environment for MBOs and mobile workflows.

Starting Sybase Unwired Platform Services

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

The way in which you start Unwired Platform services depends on the options you selected during installation. You may need to manually start Unwired Platform services.
Select **Start > Programs > Sybase > Unwired Platform > Start Unwired Platform Services**.
The Unwired Server services enable you to access the Unwired Platform runtime components and resources.

Starting Sybase Unwired WorkSpace

Start the development environment, where you can create mobile business objects (MBOs), manage EIS data sources and Unwired Server connections, develop Mobile Workflow applications, and generate Object API code.
Select **Start > Programs > Sybase > Unwired Platform > Unwired WorkSpace**.
The Sybase Unwired WorkSpace opens in the Mobile Development perspective. The Welcome page displays links to the product and information.

Next

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

Connecting to Sybase Control Center

Open the Sybase Control Center Administration Console to manage Unwired Server and its components.

From Sybase 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

For information on configuring, managing, and monitoring Unwired Server, click **Help** > **Online Documentation**.

1. Select **Start** > **Programs** > **Sybase** > **Sybase Control Center**.

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

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

   Sybase Control Center gives you access to the Unwired Platform administration features that you are authorized to use.

Learning Unwired WorkSpace Basics

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

- To access the online help, select **Help** > **Help Contents**. Some documents are for Sybase Unwired Platform, while others are for the Eclipse development environment.
- The Welcome page provides links to useful information to get you started.
  - Reopen the Welcome page by selecting **Help** > **Welcome**.
  - To close the Welcome page, click **X**.
  - To learn about tasks you must perform, select the **Development Process** icon.
- In Unwired 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 Unwired 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 Unwired Server.

Prerequisites

Complete these tasks:

1. Install Sybase Mobile SDK and Runtime as indicated in *Getting Started with Unwired Platform*.
2. Create the mobile business objects (MBOs) that you deploy to Unwired Server using one of these methods:
   - Complete *Tutorial: Mobile Business Object Development*, which provides the foundation tasks for this tutorial.
   - Download and import the completed example project if you want to bypass performing the MBO tutorial. The associated example project is available on SAP® SDN: *http://scn.sap.com/docs/DOC-8803*
3. Open the SUP101 mobile application project. In WorkSpace Navigator, right-click the **SUP101** folder and select **Open in Diagram Editor**.
4. Be sure you are using the Advanced developer profile.

Task

Note: This tutorial was developed using Mac OS X 10.7 (Lion), Xcode 4.3, and iOS SDK 5.1. If you use a different version of Xcode, 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. *Generating Object API Code*

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

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

   Register the iPhone Simulator in Sybase Control Center.
4. *Viewing the CallbackHandler File*

   In Xcode, view and understand the CallBackHandler file.
5. *Creating the User Interface*

Use Interface Builder to create and configure the user interface for the SUP101 application.

6. *Deploying the Device Application*

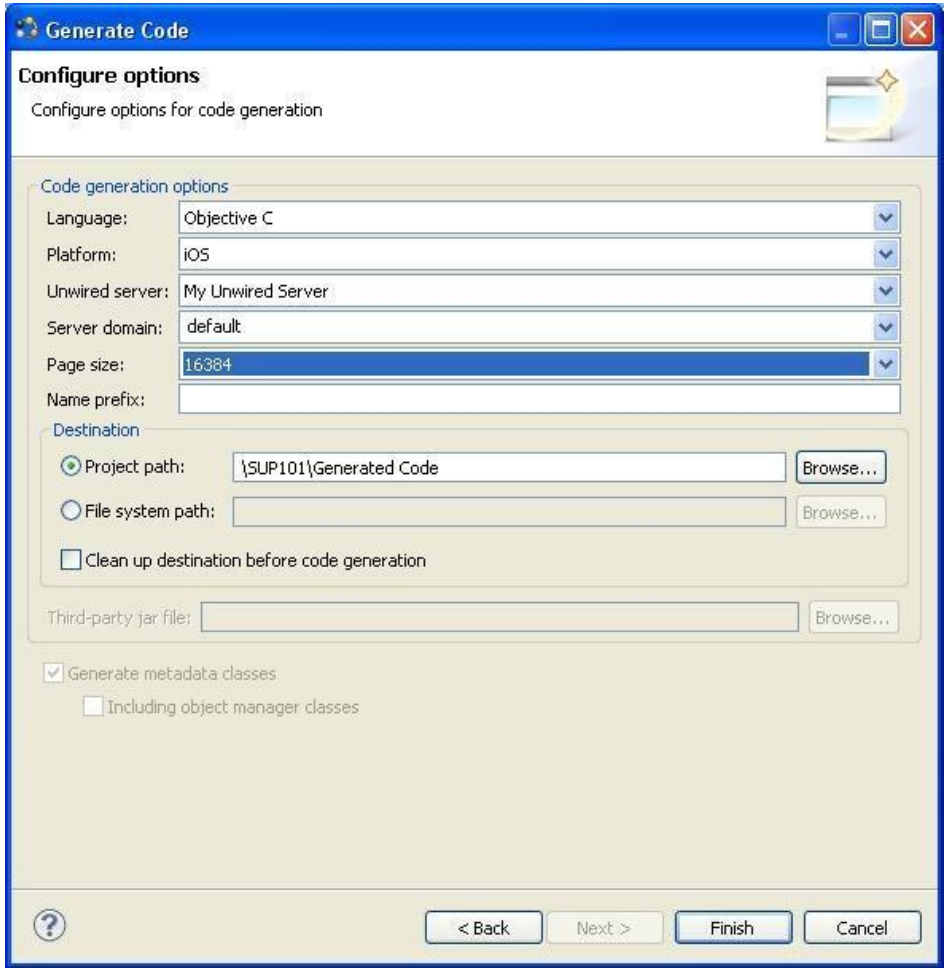
   Deploy the SUP101 application to the iPhone simulator for testing.

Generating Object API Code

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

1. In Unwired WorkSpace, open the **SUP101** mobile application project.

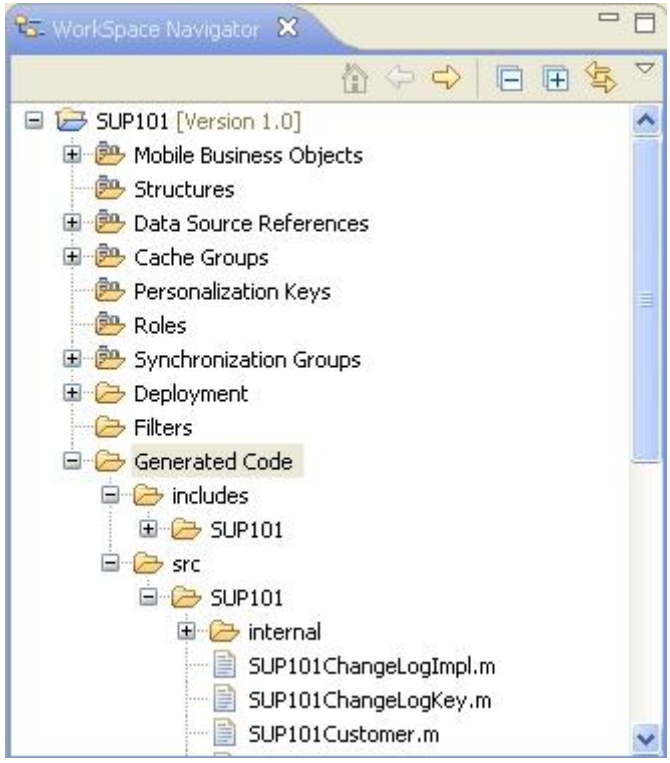
   In WorkSpace Navigator, right-click the **SUP101** folder and select **Open in Diagram Editor**.
2. Right-click in the SUP101 Mobile Application Diagram and select **Generate Code**.
3. Click **Next** in the Code Generation Configuration page.
4. In the Select Mobile Business Objects page, make sure the Customer and Sales_order MBOs are selected, then click **Next**.
5. In the Configure options page, enter these configuration options and click **Next**:



Option	Description
Language	Select **Objective C**.
Platform	Accept the default, **iOS**.
Unwired Server	Select **My Unwired Server**.
Server domain	Accept **default**. If you are not connected to Unwired Server, this field is empty. Connect to Unwired Server to proceed.
Page size	Accept the default or select a larger page size.
Name prefix	The prefix for the generated files. Leave blank.

Option	Description
Project path	Accept the default or enter a different location for the generated project files.
(Optional) Clean up destination before code generation	Select this option to delete all items in the destination folder before generating the device client files.

6. Click **Finish**, and select **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.

- Ensure the directory where Sybase Unwired Platform is installed is a shared directory so you can access it from your Mac.
- Obtain the header and Objective-C source code files you need to build the user interface from the `SUP_iOS_Custom_Dev_Tutorial_code.zip` file. This way, you can easily copy and paste the code into the corresponding files that are created in Xcode.
  - If you are viewing this guide as a PDF, you can obtain the files from the Sybase Product Documentation Web site at *http://sybooks.sybase.com/nav/summary.do?prod=1289&lang=en&submit=%A0Go%A0&prodName=Sybase+Unwired+Platform&archive=0*. Navigate to this topic in the tutorial, then click the link for the zip file to access the provided source code files.
  - If you are viewing this guide online from the Sybase Product Documentation Web site, click *SUP_iOS_Custom_Dev_Tutorial_code.zip* to access the source code files.
  - Copy the `SUP_iOS_Custom_Dev_Tutorial_code.zip` archive file to your Mac machine and extract it into a folder. It contains the Xcode project and a SUP101 project archive file to use in Unwired WorkSpace.

Task

1. On your Mac, start Xcode and select **Create a new Xcode project**.
2. Select **iOS Application** and **Single View Application** as the project template, and then click **Next**.
3. Specify these values and click **Next**.
   a) Enter `SUP101` as the **Product Name**.
   b) Enter `MyCorp` (or another value as needed) as the **Company Identifier**.
   c) Select `SUP101` for the **Class Prefix**.
   d) Select **Universal** as the **Device Family product**.
   e) Unselect **Include Unit Tests**.
   f) Unselect **Use Automatic Reference Counting**.
4. Select a location to save the project and click **Create** to open it.

   Xcode creates a folder, SUP101, to contain the project file, `SUP101.xcodeproj` and another SUP101 folder, which contains a number of automatically generated files.
5. Delete some of the automatically generated files created by default for the Xcode project.
   a) In Xcode, delete the `SUP101` folder under the SUP101 project.
      - Click **Move to Trash** instead of **Remove References**.
      - Verify the `SUP101` folder in the project folder is deleted in the Finder.
      - Verify the `SUP101.xcodeproj` is the only file in the `SUP101` folder.
6. Copy the files from the SUP101 folder on your Windows machine in to the SUP101 folder on your Mac that Xcode created to contain the SUP101 project.

a) Connect to the Microsoft Windows machine where Sybase Unwired 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 `\UnwiredPlatform\MobileSDK213\ObjectAPI\iOS` folder from the Unwired Platform installation directory to the `SUP101` folder on your Mac.
e) On your Windows machine, navigate to the SUP101 mobile application project and copy the `Generated Code` folder to the SUP101 directory on your Mac.

Next

Add libraries, resources, and source code to the SUP101 Xcode project.

See also

- *Registering the Application Connection in Sybase 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, you need to add files from the Sybase Unwired Platform folders you copied from your Windows machine.

1. In the Xcode Project Navigator, **control-click** the SUP101 folder, then select **Add Files to "SUP101"**.

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

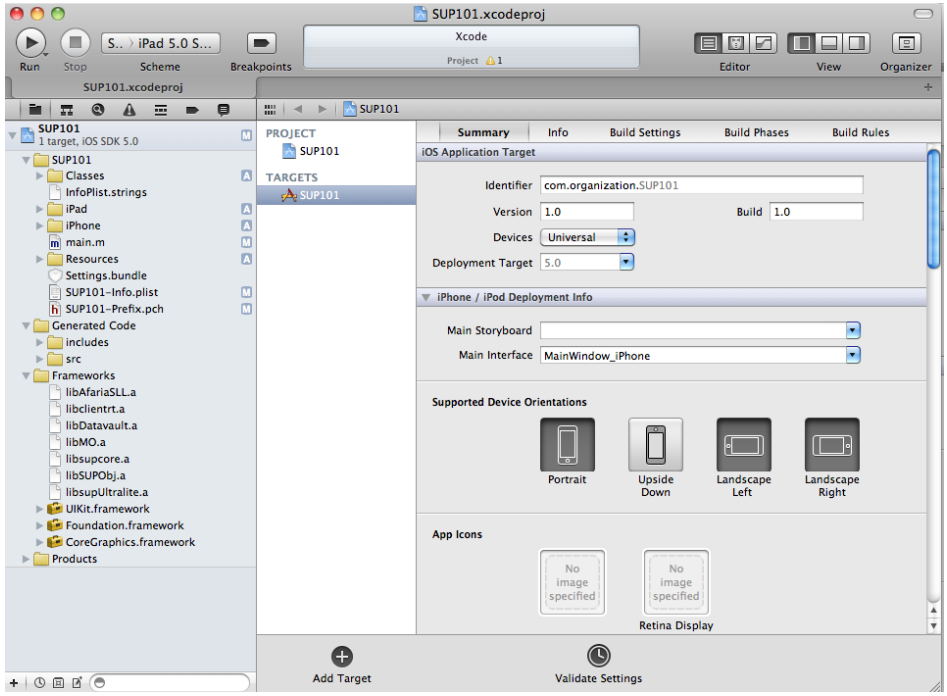
   The `Generated Code` folder is added to the project in the Project Navigator.
2. **Control-click** the **Framework** group, then select **Add Files to "SUP101"**.
   a) In the iOS folder you copied from the Sybase Unwired Platform installation, navigate to the `Libraries/Debug-iphonesimulator` directory.
   b) Select the `libclientrt.a`, `libSUPObj.a`, `libMO.a`, `libsupcore.a`, `libAfariaSLL.a`, `libDatavault.a`, and `libsupUltralite.a` libraries.
   c) Be sure **Copy items into destination group's folder (if needed)** is unselected.
   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. Add the source code files from the `SUP_iOS_Custom_Dev_Tutorial_code.zip` archive.

a) In Xcode, right-click the `SUP101` project and select **Add Files to "SUP101"**.
b) Select the **SUP101 > SUP101** folder from the SUP101 tutorial zip file.
c) Check **Copy items into destination group's folder (if needed)**.

The project now looks like this:



Next
Configure the build settings.

Configuring the Build Settings

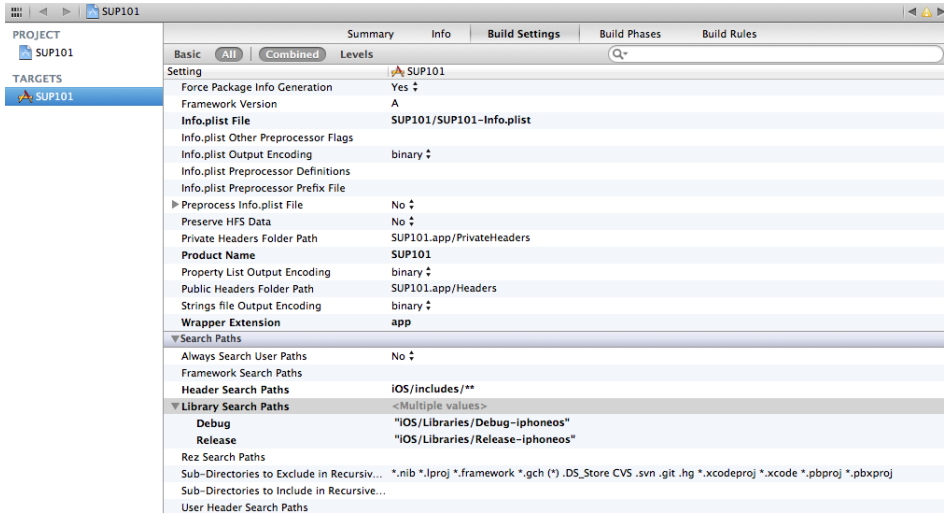
Configure the build settings for the Xcode project, then build the project.

1. In the right pane, click the **Build Settings** tab, then scroll down to the **Search Paths** section, then 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 check 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)"`. Ensure that you escape the paths using double quotes.



2. In the right pane, select the **Build Phases** tab, 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 from the SDK to the project:
   - `AddressBook.framework`
   - `CoreFoundation.framework`
   - `libicucore.A.dylib`
   - `libstdc++.dylib`
   - `libz.dylib`
   - `QuartzCore.framework`
   - `Security.framework`
   - CFNetwork.framework
   - MobileCoreServices.framework
   - SystemConfiguration.framework
3. Hold the Option key, and select **Product > Clean Build Folder**, then **Product > Build** to test the initial set up of the project. If you correctly followed this procedure, you see a **Build Succeeded** message.

Registering the Application Connection in Sybase Control Center

Register the iPhone Simulator in Sybase Control Center.

Prerequisites

Connect to Sybase Control Center.

Task

1. Log in to Sybase Control Center using the credentials you indicated during installation.
2. In Sybase Control Center, select **View > Select > Unwired 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`
   - Server name – *<localhost.sybase.com>*

     Note: The information should match the input on the client and "localhost.sybase.com" should be the actual name of your machine and domain.
   - Port – the Unwired Server port, `5001`.
   - Farm ID – `0`
   - Activation code – `123`
   - Application ID – `SUP101`
   - Domain – `default`
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 File

In Xcode, view and understand the `CallBackHandler` file.

`CallbackHandler` is a subclass of `SUPDefaultCallbackHandler`, and is used to listen for events sent from the server. It also implements the `SUPApplicationCallback` protocol to get connection, registration, and device state change notifications. The header, `CallbackHandler.h`, is referenced in a number of classes in this application, so you would create it first. You can create new Objective-C class files from the main menu: **File** > **New** > **New File**.

There are two threads involved in the SUP101 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 it is a good idea to 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.

Creating the User Interface

Use Interface Builder to create and configure the user interface for the SUP101 application.

The `SUP_iOS_Custom_Dev_Tutorial_code.zip` contains the source code for the user interface for the sample application. Although the user interface is already built once 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 32

Viewing the SubscribeController View Controller

A view controller functions as the root view screen for the SUP101 mobile application.

When you create the user interface, you assign a target action to a control object — in this example a Subscribe button so that a message (the action) is sent to another object (the target) in response to a user event, for example, a touch on the button. The view controller manages and configures the view when asked.

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

1. In the SUP101 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 MenuListController* on page 28
- *Creating the CustomerListController* on page 29
- *Adding the DetailController and Configuring the View* on page 29

Viewing the SUP101Appdelegate Files

The `SUP101Appdelegate.h` and `SUP101Appdelegate.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 SUP101Appdelegate files make use of the `SUPApplication` and `SUPDataVault` APIs to show how to store and retrieve sensitive data (such as Sybase Unwired Platform credentials) using a PIN.

The `applicationDidFinishLaunching:` method checks to see if the application has been run before, then shows a dialog to have the device user enter a PIN to unlock the application. If running for the first time, the Sybase Unwired Platform user's password is also requested.

Control passes to the `initializeSUP101` 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, it creates a new SUPDataVault secured with the user-provided PIN, to store the password and other items.
- If the application has been run before, it tries to unlock the existing vault with the provided PIN. If this fails, the application displays an error dialog and exits.

```
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 SUP101 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 SUP101 data vault using the PIN as the
password, and store the SUP 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 SUP101 data vault");
         [SUPDataVault deleteVault:kSUP101DataVaultID];
       }
  @catch(NSException *e)
  {
    // Ignore any exception
  }

  @try {
    NSLog(@"Create new SUP101 data vault and store credentials and a
generated encryption key");
    sup101vault = [SUPDataVault createVault:kSUP101DataVaultID
withPassword:self.pin withSalt:kSUP101DataVaultSalt]; // creates the
vault
    [sup101vault setString:@"password" withValue:self.SUPPassword];
    [sup101vault lock];
   }
   @catch (NSException *exception) {
     NSLog(@"Exception in creating new SUP101 data vault: %@: %@",
[exception name], [exception reason]);
   }
   @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]);
   }

}
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];
    [messagingvault unlock:self.pin withSalt:kDVStandardSalt];
  }
  @catch (NSException *exception) {
    NSLog(@"Exception unlocking messaging data vault: %@: %@",
```

```
[exception name],[exception reason]);
    [self showNoTransportAlert:kSUP101ErrorBadPin];
  }

}
```

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

```
// Start up the messaging client. This will attempt to connect to the
server. If a connection was
// established we can proceed with login. See onConnectFailure: for
more information about handling connection failure.
[[NSNotificationCenter defaultCenter] addObserver:self
selector:@selector(onConnectSuccess:) name:ON_CONNECT_SUCCESS
object:nil];
[[NSNotificationCenter defaultCenter] addObserver:self
selector:@selector(onConnectFailure:) name:ON_CONNECT_FAILURE
object:nil];
self.connectStartTime = [NSDate date];
SUPApplication* app = [SUPApplication getInstance];

@try {
  sup101vault = [SUPDataVault getVault:kSUP101DataVaultID];
  [sup101vault unlock:self.pin withSalt:kSUP101DataVaultSalt];

  app.applicationIdentifier = @"sup101";
  SUP101CallbackHandler *ch = [SUP101CallbackHandler getInstance];
  [ch retain];
  [app setApplicationCallback:ch];


  SUPConnectionProperties* props = app.connectionProperties;
  [props setServerName:self.SUPServerName];
  [props setPortNumber:[self.SUPServerPort intValue]];
  [props setUrlSuffix:@""];
  [props setFarmId:self.SUPFarmID];


  SUPLoginCredentials* login = [SUPLoginCredentials getInstance];
  if(self.SUPManualRegistration)
  {
    login.username = self.SUPConnectionName;
    login.password = nil;
    props.activationCode = self.SUPActivationCode;
  }
  else
  {
    login.username = self.SUPUserName;
    login.password = [sup101vault 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, it creates the SUP101 database, generates an encryption key, and stores it in the data vault.
- If the application has been run before, it retrieves the encryption key from the data vault, and sets it in the connection profile so the database can be used again.

```
// Normally you would not delete the local database. For this simple
example, though,
// deleting and creating an empty database will cause all data to be
sent from the
// server, and we can use [CallbackHandler onImportSuccess:] to know
when to proceed.
[SUP101SUP101DB deleteDatabase];
[SUP101SUP101DB createDatabase];
SUPConnectionProfile *cp = [SUP101SUP101DB getConnectionProfile];
[cp.syncProfile setDomainName:@"default"];
[cp enableTrace:NO];
[cp.syncProfile enableTrace:YES];

// Generate an encryption key for the database.
[SUP101SUP101DB generateEncryptionKey];
[SUP101SUP101DB closeConnection];
// Store the encryption key in the data vault for future use.
[sup101vault setString:@"encryptionkey" withValue:[cp
getEncryptionKey]];

// Since we are creating the database from scratch, we set the
encryption key for the new database

// 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, we set the
key used by the connection profile.
NSString *key = [sup101vault getString:@"encryptionkey"];
NSLog(@"Got the encryption key: %@",key);
[cp setEncryptionKey:key];
[SUP101SUP101DB closeConnection];

[SUP101SUP101DB setApplication:app];

while([app registrationStatus] != SUPRegistrationStatus_REGISTERED)
{
  NSLog(@"waiting for registration...");
  NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
  [[NSRunLoop currentRunLoop] runUntilDate:[NSDate
dateWithTimeIntervalSinceNow:1]];
  [pool release];
}
while([app connectionStatus] != SUPConnectionStatus_CONNECTED)
{
  NSLog(@"waiting for connection...");
  NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
  [[NSRunLoop currentRunLoop] runUntilDate:[NSDate
dateWithTimeIntervalSinceNow:1]];
  [pool release];
}
```

Once the server connection is made, the `onConnectionStatusChanged:` method in the callback handler posts an ON_CONNECT_SUCCESS notification, and the SUP101AppDelegate's `onConnectSuccess:` method is called. This method performs a database subscribe to the Unwired Server, using the username from the application settings, and the password from the data vault.

If you are connecting to the Unwired Server through a Relay Server then you must provide more information for the database synchronization profile:

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

```
SUPConnectionProfile *sp = [SUP101SUP101DB
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"];
```

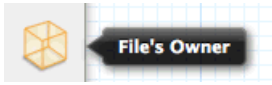
  - **NetworkProtocol –** http or https.
  - **PortNumber –** the correct port number for the selected NetworkProtocol.
  - **NetworkStreamParams –** certificateName: the name of the certificate you added in the Resource folder.

    urlsuffixProvidedByTheRelayServer: the URL suffix provided by the Relay Server

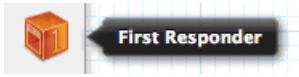
Configuring the SubscribeController View

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.

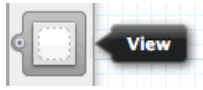
1. Click the `SubscribeController.xib` file to reveal a view of the (presently empty) screen in the right pane and the following three items represented by icons in the middle pane:
   - **File's Owner** – the object that is set to be the owner of the user interface, which is typically the object that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

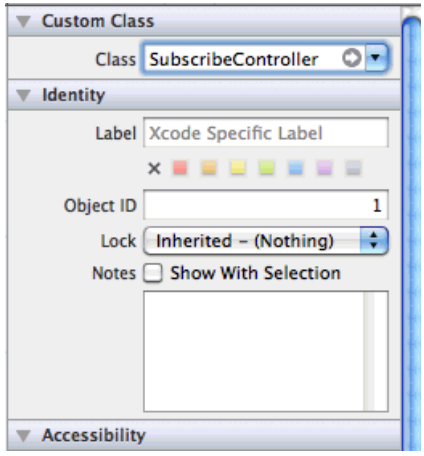
   - **First Responder** – the first responder proxy object handles events. Connecting an action to the first responder means that when the action is invoked, it is dynamically sent to the responder chain.

- **View** – displayed in a separate window to allow you to edit it.

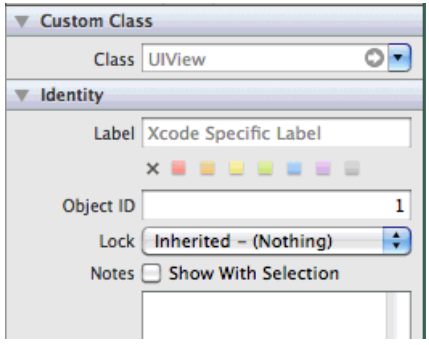


2. Select the **File's Owner** icon, click **View** in the utility area, click **Show the Identity Inspector**, and make sure `SubscribeController` appears in the **Class** field under **Custom Class**.

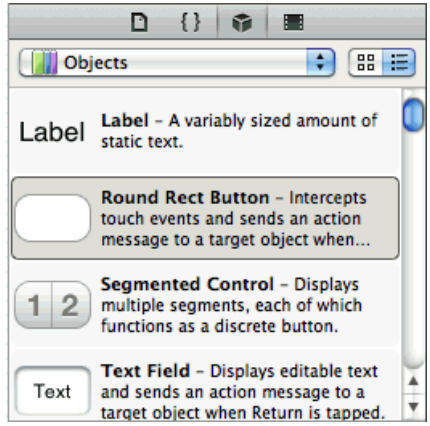


This tells Interface Builder the class of the object to allow you to make connections to and from the File's Owner.

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



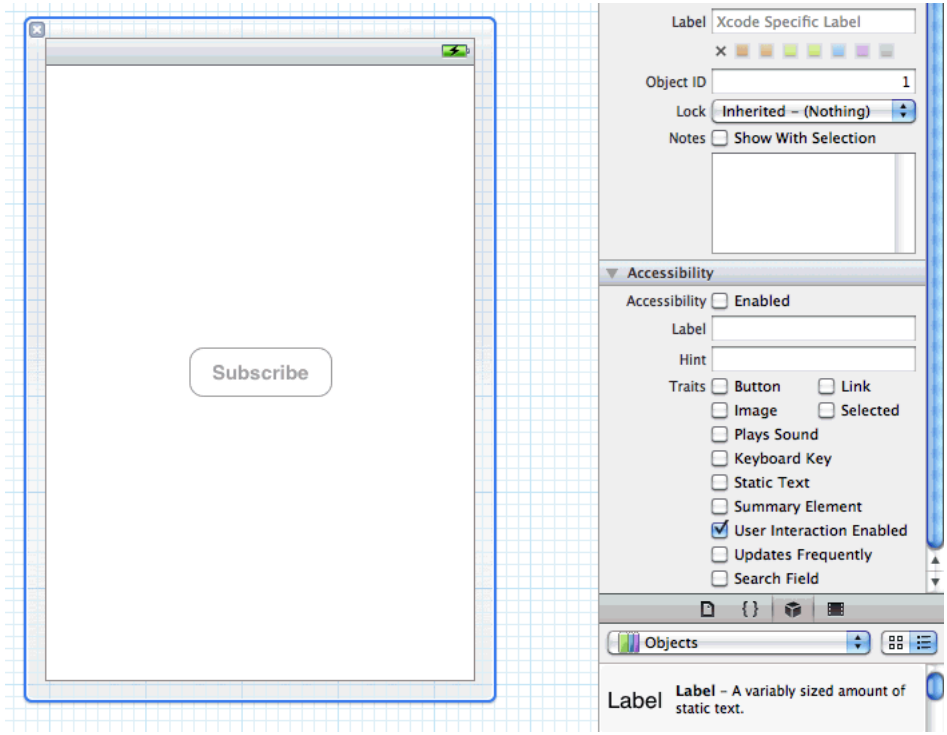
4. To create a Subscribe button, select **View > Utilities > Show Object Library**.
   a) In the Object Library panel, select the **Round Rect Button** item, drag it onto the view.



b) Double-click it and enter `Subscribe` and press **Return**.

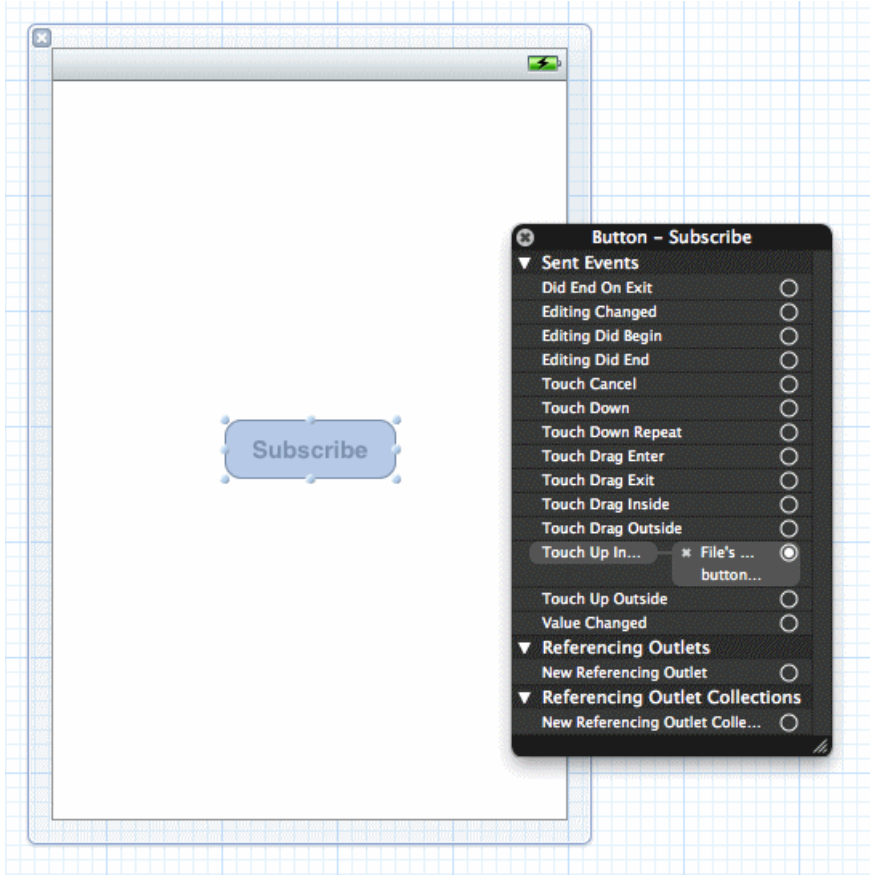
5. In the Accessibility section of the Identity Inspector, make sure **Enabled** is unselected.

   Disable the button because the application cannot subscribe to the server for updates until it is connected.



6. Control-click the **Subscribe** button to show the inspector.

7. Drag from the circle to the right of **Touch Up Inside** to the **File's Owner** icon and release, then click on **buttonPressed** to establish a connection between the **Subscribe** button and the button's action method:

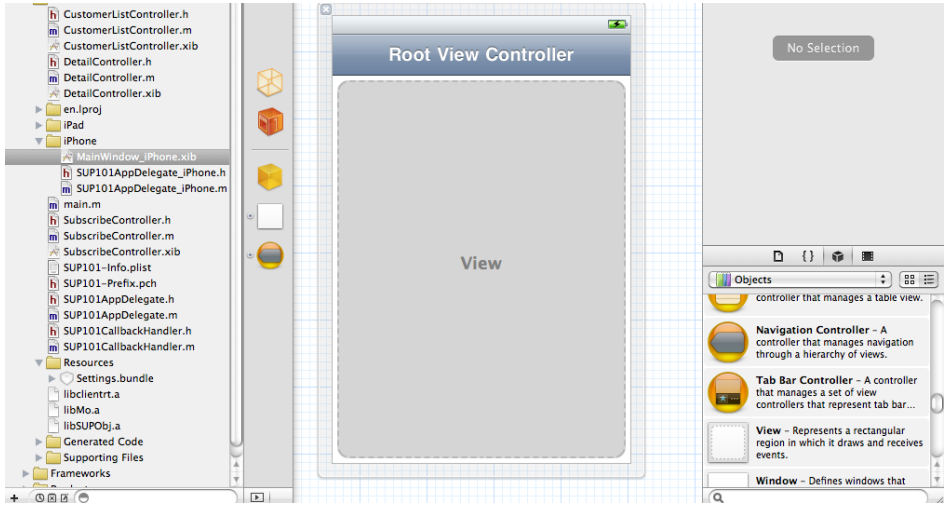


Making Connections

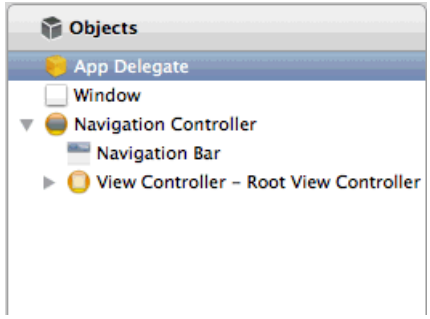
Add Navigation Controllers to `MainWindow_iPhone.xib` and `MainWindow_iPad.xib` and create a connection from the AppDelegate to the Navigation Controller.

1. In the left pane, under the iPhone folder, click the `MainWindow_iPhone.xib` file.

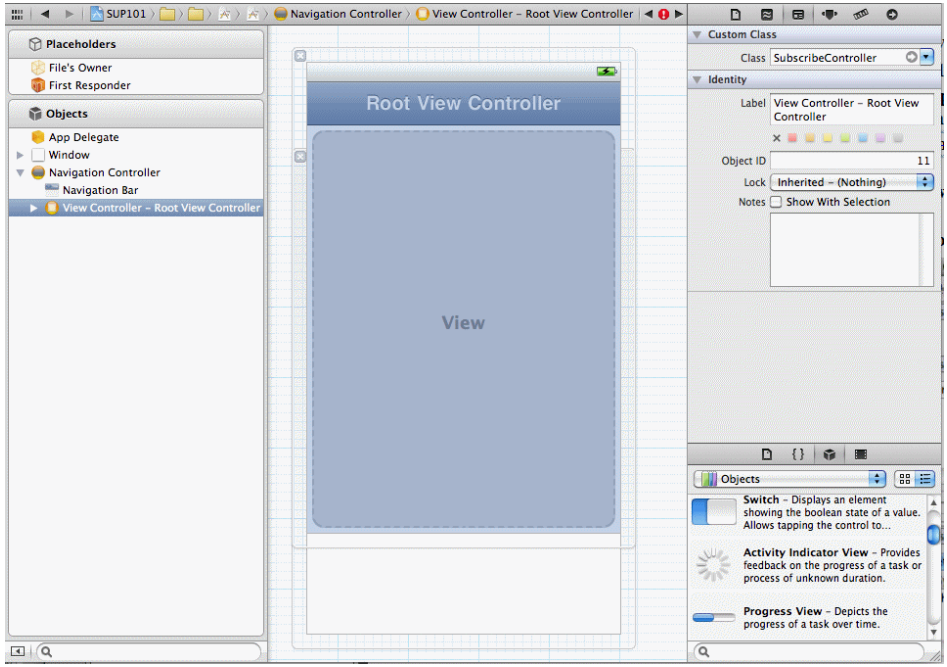
   If you do not see the Navigation Controller in the middle pane, drag it from **Objects** to the middle pane:



2. Under **Objects** in the middle pane, control-drag from the **AppDelegate** icon to the **Navigation Controller** icon to create a **navController** outlet.



3. Click on the expansion arrow at the bottom of the middle pane to switch to list view, select **Navigation Controller** > **View Controller**, and in the Identity Inspector, select `SubscribeController` in the **Class** field.



Once the class is selected, the **ViewController** name in the hierarchy changes to **SubscribeController** and the connection from the AppDelegate to the Navigation Controller is created.

4. Repeat these steps to add a navigation controller to `MainWindow_iPad.xib`.

Creating the MenuListController

Create the menu list view.

The source files you added from the `SUP_iOS_Custom_Dev_Tutorial_code.zip` file 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`. Ensure that you indicate **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 three menu items: List, Create, and Find By PrimaryKey. Tap a row to move to the corresponding screen.

See also

- *Viewing the SubscribeController View Controller* on page 18

Creating the CustomerListController

Create the customer list view.

The source files you added from the SUP_iOS_Custom_Dev_Tutorial_code.zip file contain the `CustomerListController.h`, `CustomerListController.m`, and `CustomerListController.xib` files that create the customer list view. 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 `UITableViewController`. Be sure to indicate **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 data source for displaying the table view.

   If a row is tapped, the `accessoryButtonTappedForRowWithIndexPath` method is run, 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 18
- *Creating the MenuListController* on page 28
- *Adding the DetailController and Configuring the View* on page 29

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 SUP_iOS_Custom_Dev_Tutorial_code.zip file 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`. Be sure to indicate **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** > **Object Library**.
3. In the Object Library panel, select the **Text Field** item, 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, 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 to `Submit`. Also 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 "SUP101Customer.h"
#import "SUPCallbackHandler.h"
#import "CallbackHandler.h"



@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) SUP101Customer *originalObj;
@property (nonatomic, retain) IBOutlet UIButton *submitButton;
@property (nonatomic, retain) IBOutlet UIButton *deleteButton;
@property (nonatomic, retain) IBOutlet UILabel *label;
@property (nonatomic, retain) CallbackHandler *callbackHandler;


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


@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.

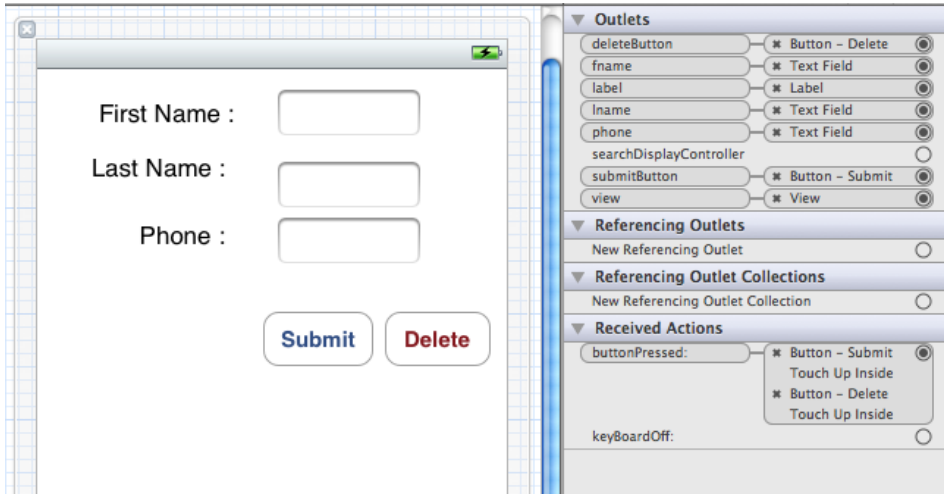
If the server accepts the changes, the callback handler posts an ON_REPLAY_SUCCESS notification, which causes the `onReplaySuccess` notification handler to run. The cached UI data is refreshed from the database and the Submit button is re-enabled.

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 and then initiates a synchronize 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 will be posted.

7. Click the `DetailController.xib` file to open it in Interface Builder, click the **First Name** text field, and select **View** > **Utilities** > **Attributes Inspector**.
8. In the Attributes Inspector panel , 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. Control-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** > **Connections Inspector** to confirm that the outlets have been correctly configured:



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

See also

- *Viewing the SubscribeController View Controller* on page 18
- *Creating the MenuListController* on page 28
- *Creating the CustomerListController* on page 29

Deploying the Device Application

Deploy the SUP101 application to the iPhone simulator for testing.

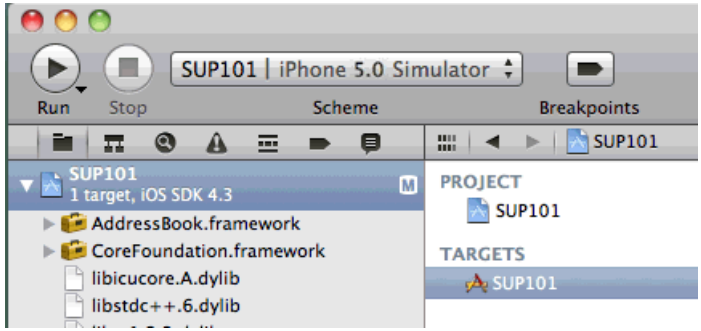
Prerequisites

Register an application connection in Sybase Control Center.

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

Task

1. In the upper-left corner of Xcode, make sure that the **Scheme** is set to **SUP101 | iPhone 5.0 Simulator**.



2. Select **Product** > **Build** and then **Product** > **Run**

   The project is built and the iPhone simulator starts.
3. In the iPhone applications screen, open the **SUP101** 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. These settings include the server name, port number, Sybase Unwired Platform user name, and other settings.
4. In the iPhone simulator, go to **Settings** > **SUP101** to enter the connection settings.
   - **SUP Server** – the machine that hosts the server where the SUP101 mobile application project is deployed.
   - **SUP Server Port** – Unwired Server port number. The default is **5001**.
   - **Farm ID** – the company ID you entered when you registered the device in Sybase Control Center, in this case, `0`.
   - **SUP Username** – the user to be authenticated, `supAdmin`.


Carrier
5:16 PM
Settings
SUP101
SUP 101 Settings
SUP Server dlowder-w7vm
SUP Server Port 5001
Farm ID 0
SUP Username supAdmin
Manual registration ON
If manual connection:
Connection Name user1
Activation Code ●●●

If the "Manual registration" switch is set to "Off", the application will attempt to perform an automatic registration, creating an application registration with the same name as the SUP Username ("supAdmin" in this example). This feature allows a client with a valid SUP username and password to connect and register with the server without a need for a manual registration to be created in advance.

If the "Manual registration" switch is set to "On", then the connection name and activation code need to be filled in, and must match an application connection that has already been created in Sybase Control Center (see *Register an application connection in Sybase Control Center*).

5. In the iPhone applications screen, reopen the **SUP101** application.
   You are prompted for a PIN to use to securely store your Sybase Unwired Platform password, and a database encryption key that is generated when the application launches. For subsequent launches of the application, only the PIN is required.
6. Enter a PIN, and enter the password for the SUP username that was entered in step 4.
7. Click **Synchronization**.
   The menu appears.
8. Click **List**.
   The customer list appears.
9. Select a customer record from the customer list and double-click to open the detail view.
   The customer detail shows the fields: **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 18

Learn More About Sybase Unwired Platform

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

Check the Sybase Product Documentation Web site regularly for updates: *http://sybooks.sybase.com/sybooks/sybooks.xhtml*, 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 is the end results of a finished tutorial without going through the steps. 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 Sybase Unwired 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 Sybase Unwired 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.

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: Mobile Workflow Packages*

Customize and automate:

- *Developer Guide: Unwired Server Management API* – customize and automate system administration features.

Javadoc and HeaderDoc are also available in the installation directory.

Index