Version 27.0: Spring '13

Mobile SDK Development Guide

Salesforce.com Mobile Development

Last updated: February 15, 2013

Table of Contents

Chapter 1

Introduction to Mobile Development

In this chapter ...

Force.com has proven itself as an easy, straightforward, and highly productive platform for cloud computing. Developers can define application components, such as custom objects and fields, workflow rules, Visualforce pages, and Apex classes and triggers, using point-and-click tools of the Web interface, and assembling the components into killer apps. As a mobile developer, you might be wondering how you can leverage the power of the Force.com platform to create sophisticated apps.

The Mobile SDK provides essential libraries for quickly building native or hybrid mobile apps that seamlessly integrate with the Force.com cloud architecture, and simplifies development by providing:

- OAuth implementations that work out-of-the-box
- OAuth access token management, including persistence and refresh capabilities
- Native REST API wrappers
- Containers for building hybrid applications

Note:

The Mobile SDK Developer Guide is currently a Beta release. The full guide will be available with the version 2.0 release of the SDK in early 2013. For walk-through tutorials, download the Mobile SDK Workbook from https://github.com/forcedotcom/SalesforceMobileSDK-Samples. The workbook goes through each development scenario, albeit in less detail than this guide. Also, be sure to visit Force.com regularly for blog postings and other updates.

Intended Audience

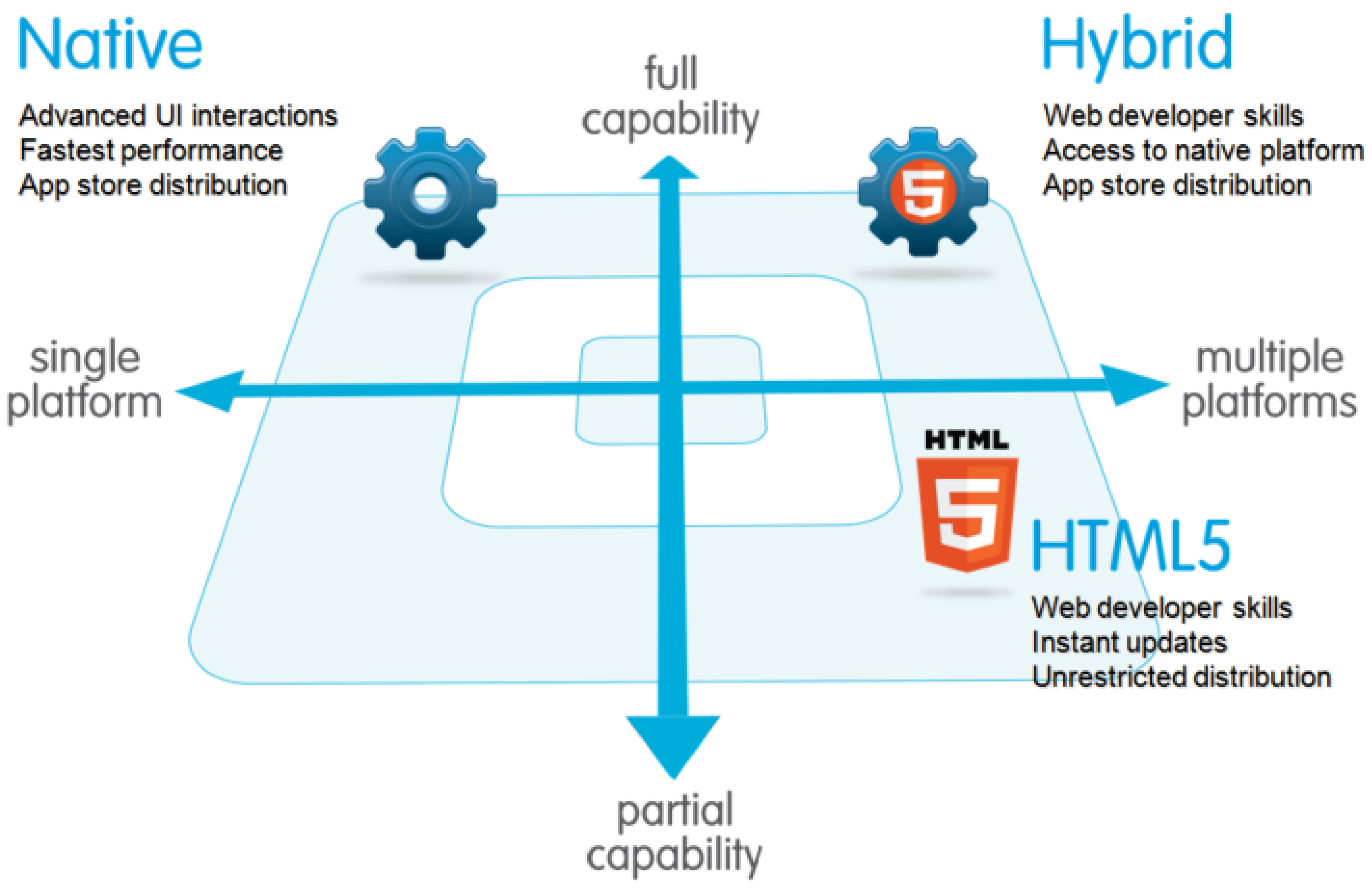
This guide is primarily for developers who are already familiar with mobile technology, such as OAuth2 and the REST API, and who probably have some Force.com experience. But if that doesn't exactly describe you, don't worry. We've tried to make this guide usable by a wider audience. For example, you might be a Salesforce admin tasked with developing a new mobile app to support your organization, or you might be a mobile developer who's entirely new to Force.com. If either of those descriptions fit you, then you should be able to follow along just fine.

If you're an admin setting up users for mobile devices or mobilizing existing functionality, you're probably looking for the Salesforce Mobile Implementation Guide.

About Native, HTML5, and Hybrid Development

Many factors play a part in your mobile strategy, such as your team's development skills, required device functionality, the importance of security, offline capability, interoperability, and so on. In the end, it's not just a question of what your app will do, but how you'll get it there. The Mobile SDK offers three ways to create mobile apps:

- **Native** apps are specific to a given mobile platform (iOS or Android) and use the development tools and language that the respective platform supports (e.g., Xcode and Objective-C with iOS, Eclipse and Java with Android). Native apps look and perform the best but require the most development effort.
- **HTML5** apps use standard web technologies—typically HTML5, JavaScript and CSS—to deliver apps through a mobile Web browser. This "write once, run anywhere" approach to mobile development creates cross-platform mobile applications that work on multiple devices. While developers can create sophisticated apps with HTML5 and JavaScript alone, some challenges remain, such as session management, secure offline storage, and access to native device functionality (camera, calendar, geolocation, etc.)
- **Hybrid** apps combine the ease of HTML5 Web app development with the power of the native platform by wrapping a Web app inside the Salesforce Mobile Container. This combined approach produces an application that can leverage the device's native capabilities and be delivered through the app store. You can also create hybrid apps using Visualforce pages delivered through the native container.



Native Apps

Native apps provide the best usability, the best features, and the best overall mobile experience. There are some things you get only with native apps:

- **Multi touch**—double taps, pinch-spread, and other compound UI gestures.
- **Fast graphics API**—the native platform gives you the fastest graphics, which might not be a big deal if you're showing a static screen with only a few elements, or a very big deal if you're using a lot of data and require a fast refresh.
- **Fluid animation**—related to the fast graphics API is the ability to have fluid animation. This is especially important in gaming, highly interactive reporting, or intensely computational algorithms for transforming photos and sounds.
- **Built-in components**—The camera, address book, geolocation, and other features native to the device can be seamlessly integrated into mobile apps. Another important built-in component is encrypted storage, but more about that later.
- **Ease of use**—The native platform is what people are accustomed to. When you add that familiarity to the native features they expect, your app becomes that much easier to use.

Native apps are usually developed using an integrated development environment (IDE). IDEs provide tools for building, debugging, project management, version control, and other tools professional developers need. You need these tools because native apps are more difficult to develop. Likewise, the level of experience required is higher than in other development scenarios. If you're a professional developer, you don't have to be sold on proven APIs and frameworks, painless special effects through established components, or the benefits of having all your code in one place.

HTML5 Apps

An HTML5 mobile app is basically a web page, or series of web pages, that are designed to work on a small mobile device screen. As such, HTML5 apps are device agnostic and can be opened with any modern mobile browser. Because your content is on the web, it's searchable, which can be a huge benefit for certain types of apps (shopping, for example).

If you're new to mobile development, the technological bar is lower for Web apps; it's easier to get started here than in native or hybrid development. Unfortunately, every mobile device seems to have its own idea of what constitutes usable screen size and resolution. This diversity imposes an additional burden of testing on different devices. Browser incompatibility is especially common on Android devices, for example.

An important part of the "write once, run anywhere" HTML5 methodology is that distribution and support is much easier than for native apps. Need to make a bug fix or add features? Done and deployed for all users. For a native app, there are longer development and testing cycles, after which the consumer typically must log into a store and download a new version to get the latest fix.

If HTML5 apps are easier to develop, easier to support, and can reach the widest range of devices, where do these apps lose out?

- **Offline storage** — You can implement a semblance of offline capability by caching files on the device. Even if the underlying database is encrypted, this is not as secure or as well segmented as a native keychain encryption that protects each app with a developer certificate.
- **Security** — In general, implementing even trivial security measures on a native platform can be complex tasks for a mobile Web developer. It can also be painful for users. For example, if a web app with authentication is launched from the desktop, it will require users to enter their credentials every time the app is sent to the background.
- **Native features** — the camera, address book, and other native features are accessible on limited, if any, browser platforms.
- **Native look and feel** — HTML5 can only emulate the native look, while customers won't be able to use familiar compound gestures.

Hybrid Apps

Hybrid apps are built using HTML5 and JavaScript wrapped inside a thin container that provides access to native platform features. For the most part, hybrid apps provide the best of both worlds, being almost as easy to develop as HTML5 apps with all the functionality of native.

You know that native apps are installed on the device, while HTML5 apps reside on a Web server, so you might be wondering whether hybrid apps store their files on the device or on a server? The answer is: both. You can implement a hybrid app two ways.

Local

You can package HTML and JavaScript code inside the mobile application binary, in a manner similar to the structure of a native application. In this scenario you use REST APIs and Ajax to move data back and forth between the device and the cloud.

Server

Alternatively, you can implement the full web application from the server (with optional caching for better performance). Your container app retrieves the full application from the server and displays it in a browser window.

Both types of hybrid development are covered in this guide.

Native, HTML5, and Hybrid Summary

The following table sums up how the three mobile development scenarios stack up.

	Native	HTML5	Hybrid
Graphics	Native APIs	HTML, Canvas, SVG	HTML, Canvas, SVG

	Native	**HTML5**	**Hybrid**
Performance	Fast	Slow	Slow
Look and feel	Native	Emulated	Emulated
Distribution	App store	Web	App store
Camera	Yes	Browser dependent	Yes
Notifications	Yes	No	Yes
Contacts, calendar	Yes	No	Yes
Offline storage	Secure file system	Shared SQL	Secure file system, shared SQL
Geolocation	Yes	Yes	Yes
Swipe	Yes	Yes	Yes
Pinch, spread	Yes	Yes	Yes
Connectivity	Online, offline	Mostly online	Online, offline
Development skills	Objective C, Java	HTML5, CSS, JavaScript	HTML5, CSS, JavaScript

Enough Talk; I'm Ready

If you'd rather read about the details later, there are Quick Start topics in this guide for each native development scenario. There's also a Mobile SDK Workbook that demonstrates downloading and running a simple mobile app.

- iOS Native Quick Start
- Android Native Quick Start
- Download the Mobile SDK Workbook from https://github.com/forcedotcom/SalesforceMobileSDK-Samples

Development Prerequisites

It's helpful to have some experience with Database.com or Force.com. You'll need either a Database.com account or a Force.com Developer Edition organization.

This guide also assumes you are familiar with the following platforms and technologies:

- To build iOS applications, you'll need Mac OS X Snow Leopard or Lion, and Xcode 4.2+.
- To build Android applications, you'll need the Java JDK 6, Eclipse, Android ADT plugin, and the Android SDK.
- To build hybrid applications, you'll need an organization that has Visualforce.
- Most of our resources are on GitHub, a social coding community. You can access all of our files in our public repository, but we think it's a good idea to join. https://github.com/forcedotcom

Choosing Between Database.com and Force.com

You can build mobile applications that store data on a Database.com or Force.com organization. Hereafter, this guide assumes you are using a Force.com Developer Edition that uses Force.com login end points such as `login.salesforce.com`. However, you can simply substitute your Database.com credentials in the appropriate places.

Note: If you choose to use Database.com, you can't develop Visualforce–driven hybrid apps.

Sign Up for Force.com

1. In your browser go to `developer.force.com/join`.
2. Fill in the fields about you and your company.
3. In the `Email Address` field, make sure to use a public address you can easily check from a Web browser.
4. Enter a unique `Username`. Note that this field is also in the *form* of an email address, but it does not have to be the same as your email address, and in fact, it's usually better if they aren't the same. Your username is your login and your identity on `developer.force.com`, and so you're often better served by choosing a username that describes the work you're doing, such as `develop@workbook.org`, or that describes you, such as `firstname@lastname.com`.
5. Read and then select the checkbox for the `Master Subscription Agreement`.
6. Enter the Captcha words shown and click **Submit Registration**.
7. In a moment you'll receive an email with a login link. Click the link and change your password.

Sign Up for Database.com

1. In your browser go to `www.database.com`.
2. Click **Signup**.
3. Fill in the fields about you and your company.
4. In the `Email Address` field, make sure to use a public address you can easily check from a Web browser.
5. The `Username` field is also in the *form* of an email address, but it does not have to be the same as your actual email address, or even an email that you use. It's helpful to change the username to something that describes the use of the organization. In this workbook we'll use admin-user@workbook.db.
6. Enter the Captcha words shown.
7. Read and then select the checkbox for the `Master Subscription Agreement` and supplemental terms.
8. Click **Sign Up**.
9. After signing up, you'll be sent an email with a link that you must click to verify your account. Click the link.
10. Now supply a password, and a security question and answer.

Keeping Up With the Mobile SDK

The Mobile SDK evolves rapidly, so you'll want to check the following regularly.

- You can always find the most current releases in our Mobile SDK GitHub Repository
- Keep up to date with What's New.

- The latest articles, blog posts, tutorials, and webinars are on http://wiki.developerforce.com/page/Mobile_SDK.
- Join the conversation on our message boards at http://boards.developerforce.com/t5/Mobile/bd-p/mobile.

Mobile SDK GitHub Repository

The Mobile SDK development team uses GitHub to host and store source code for the Mobile SDK. You don't need to sign up for GitHub to access the Mobile SDK, but we think it's a good idea to be part of this social coding community. https://github.com/forcedotcom

You can always find the latest Mobile SDK releases in our public repositories:

- `https://github.com/forcedotcom/SalesforceMobileSDK-iOS`
- `https://github.com/forcedotcom/SalesforceMobileSDK-Android`
- `https://github.com/forcedotcom/SalesforceMobileSDK-Shared`

Note: You might want to bookmark the Mobile SDK home page http://wiki.developerforce.com/page/Mobile_SDK, for the latest articles, blog posts, tutorials, and webinars.

What's New in This Release

For a summary of what's new and changed in this release of the Salesforce Mobile SDK, visit the Mobile SDK Release Notes. This page also provides a history of previous releases.

Chapter 2

Authentication, Security, and Identity in Mobile Apps

In this chapter ...

Secure authentication is essential for enterprise applications running on mobile devices. OAuth2 is the industry-standard protocol that allows secure authentication for access to a user's data, without handing out the username and password. It is often described as the valet key of software access: a valet key only allows access to certain features of your car: you cannot open the trunk or glove compartment using a valet key.

The Salesforce OAuth2 implementation can quickly and easily be embedded by mobile app developers. The implementation uses an HTML view to collect the username and password, which are then sent to the server. A session token is returned and securely stored on the device for future interactions.

Remote Access and OAuth Terminology

Access Token

A value used by the consumer to gain access to protected resources on behalf of the user, instead of using the user's Salesforce credentials. The access token is a session ID, and can be used directly.

Authorization Code

A short-lived token that represents the access granted by the end user. The authorization code is used to obtain an access token and a refresh token.

Consumer Key

A value used by the consumer to identify itself to Salesforce. Referred to as `client_id`.

Refresh Token

A token used by the consumer to obtain a new access token, without having the end user approve the access again.

Remote Access Application

A *remote access application* is an application external to Salesforce that uses the OAuth protocol to verify both the Salesforce user and the external application.

Creating a Remote Access Application

Before a mobile device can connect with the service, you'll need to create a remote access application. The remote access application includes a Consumer Key, a prerequisite to all development scenarios in this guide.

1. Log into your Database.com or Force.com instance.
2. Navigate to **App Setup** > **Develop** > **Remote Access**.
3. Click **New**.
4. For `Application`, enter a name, such as `Test Client`
5. For `Callback URL`, enter `sfdc://success`

   Note: The `Callback URL` does not have to be a valid URL; it only has to match what the app expects in this field. You can use any custom prefix, such as `sfdc://`.

6. For `Email`, enter your email address.
7. Click **Save**.

Tip: The detail page for your remote access configuration displays a consumer key. It's a good idea to copy the key, as you'll need it later.

OAuth2 Authentication Flow

The authentication flow depends on the state of authentication on the device.

First Time Authentication Flow

1. User opens a mobile application.
2. An authentication dialog/window/overlay appears.
3. User enters username and password.
4. App receives session ID.
5. User grants access to the app.
6. App starts.

Ongoing Authentication

1. User opens a mobile application.
2. If the session ID is active, the app starts immediately. If the session ID is stale, the app uses the refresh token from its initial authorization to get an updated session ID.
3. App starts.

PIN Code Authentication

1. User opens a mobile application after not using it for some time.
2. If the elapsed time exceeds the configured PIN timeout value, a PIN entry screen appears. User enters the PIN.

   Note: PIN is a function of the mobile policy - it can be shown whether you are online or offline, if enough time has elapsed since you last used the application.

3. App re-uses existing session ID.
4. App starts.

OAuth 2.0 User-Agent Flow

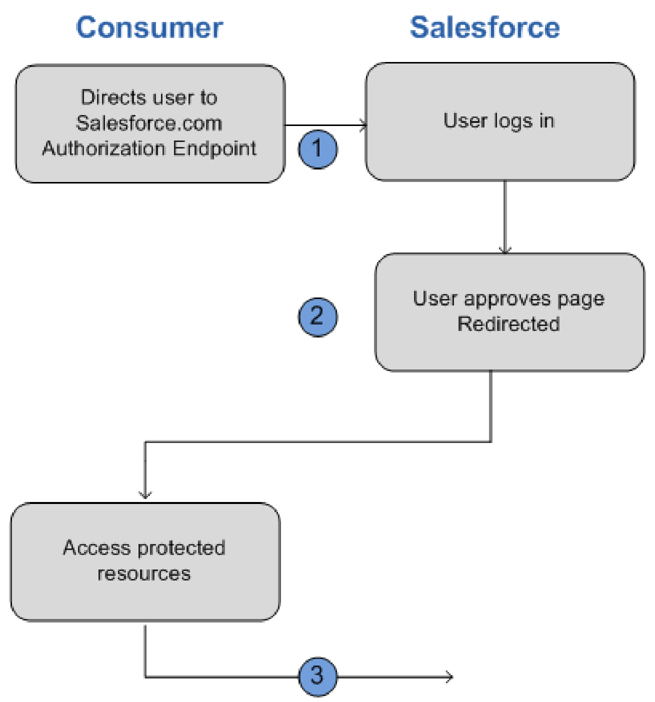
The user-agent authentication flow is used by client applications residing on the user's mobile device. The authentication is based on the user-agent's same-origin policy.

In the user-agent flow, the client application receives the access token in the form of an HTTP redirection. The client application requests the authorization server to redirect the user-agent to another web server or local resource accessible to the user-agent, which is capable of extracting the access token from the response and passing it to the client application. Note that the token response is provided as a hash (#) fragment on the URL. This is for security, and prevents the token from being passed to the server, as well as to other servers in referral headers.

This user-agent authentication flow doesn't utilize the client secret since the client executables reside on the end-user's computer or device, which makes the client secret accessible and exploitable.

Warning: Because the access token is encoded into the redirection URI, it might be exposed to the end-user and other applications residing on the computer or device.

If you are authenticating using JavaScript, call `window.location.replace();` to remove the callback from the browser's history.


Consumer
Salesforce
Directs user to Salesforce.com Authorization Endpoint
1
User logs in
2
User approves page Redirected
Access protected resources
3

1. The client application directs the user to Salesforce to authenticate and authorize the application.
2. The user must always approve access for this authentication flow. After approving access, the application receives the callback from Salesforce.

After a consumer has an access token, they can use the access token to access data on the end-user's behalf and receive a refresh token to get a new access token if it becomes invalid for any reason.

OAuth 2.0 Refresh Token Flow

After the consumer has been authorized for access, they can use a refresh token to get a new access token (session ID.) This is only done after the consumer already has received a refresh token using either the Web server or user-agent flow. It is up to the consumer to determine when an access token is no longer valid, and when to apply for a new one. Bearer flows can only be used after the consumer has received a refresh token.

The following are the steps for the refresh token authentication flow. More detail about each step follows:

1. The consumer uses the existing refresh token to request a new access token.
2. After the request is verified, Salesforce sends a response to the client.

Note:

SmartStore data is inherently volatile. Its lifespan is tied to the authenticated user as well as to OAuth token states. When the user logs out of the app, SmartStore deletes all soup data associated with that user. Similarly, when the OAuth refresh token is revoked or expires, the user's app state is reset, and all data in SmartStore is purged. Carefully consider the volatility of SmartStore data when designing your app. This warning is especially important if your org sets a short lifetime for the refresh token.

Scope Parameter Values

The `scope` parameter enables you to fine-tune what the client application can access in a Salesforce organization. The valid values for `scope` are:

Value	Description
`api`	Allows access to the current, logged-in user's account over the APIs, such as REST API or Bulk API. This also includes `chatter_api`, allowing access to Chatter API resources.
`chatter_api`	Allows access to only the Chatter API resources.
`full`	Allows access to all data accessible by the logged-in user. `full` does not return a refresh token. You must explicitly request the `refresh_token` scope to get a refresh token.
`id`	Allows access only to the identity URL service.
`refresh_token`	Allows a refresh token to be returned if you are eligible to receive one.
`visualforce`	Allows access to Visualforce pages.
`web`	Allows the ability to use the `access_token` on the Web. This also includes `visualforce`, allowing access to Visualforce pages.

Using Identity URLs

In addition to the access token, an identity URL is also returned as part of a token response, in the `id` parameter.

The identity URL is both a string that uniquely identifies a user, as well as a RESTful API that can be used to query (with a valid access token) for additional information about the user. Salesforce returns basic personalization information about the user, as well as important endpoints that the client can talk to, such as photos for the user, and API endpoints it can access.

The format of the URL is: `https://login.salesforce.com/id/`*`orgID`*`/`*`userID`*, where *`orgId`* is the ID of the Salesforce organization that the user belongs to, and *`userID`* is the Salesforce user ID.

Note: For Sandbox, `login.salesforce.com` is replaced with `test.salesforce.com`.

The URL must always be HTTPS.

Identity URL Parameters

The following parameters can be used with the access token and identity URL. They are used in an authorization request header or in a request with the `oauth_token` parameter. For more details, see "Using the Access Token" in the online help.

Parameter	Description
Access token	See "Using the Access Token" in the online help.
Format	This parameter is optional. Specify the format of the returned output. Valid values are: • `urlencoded` • `json`

Parameter	Description
	• `xml` Instead of using the `format` parameter, the client can also specify the returned format in an accept-request header using one of the following: • `Accept: application/json` • `Accept: application/xml` • `Accept: application/x-www-form-urlencoded` Note the following: • Wildcard accept headers are allowed. `*/*` is accepted and returns JSON. • A list of values is also accepted and is checked left-to-right. For example: `application/xml,application/json,application/html,*/*` returns XML. • The `format` parameter takes precedence over the accept request header.
Version	This parameter is optional. Specify a SOAP API version number, or the literal string, `latest`. If this value isn't specified, the returned API URLs contains the literal value `{version}`, in place of the version number, for the client to do string replacement. If the value is specified as `latest`, the most recent API version is used.
PrettyPrint	This parameter is optional, and is only accepted in a header, not as a URL parameter. Specify the output to be better formatted. For example, use the following in a header: `X-PrettyPrint:1`. If this value isn't specified, the returned XML or JSON is optimized for size rather than readability.
Callback	This parameter is optional. Specify a valid JavaScript function name. This parameter is only used when the format is specified as JSON. The output is wrapped in this function name (JSONP.) For example, if a request to `https://server/id/orgid/userid/` returns `{"foo":"bar"}`, a request to `https://server/id/orgid/userid/?callback=baz` returns `baz({"foo":"bar"});`.

Identity URL Response

After making a valid request, a **302 redirect** to an instance URL is returned. That subsequent request returns the following information in JSON format:

- `id`—The identity URL (the same URL that was queried)
- `asserted_user`—A boolean value, indicating whether the specified access token used was issued for this identity
- `user_id`—The Salesforce user ID
- `username`—The Salesforce username

- `organization_id`—The Salesforce organization ID
- `nick_name`—The community nickname of the queried user
- `display_name`—The display name (full name) of the queried user
- `email`—The email address of the queried user
- `status`—The user's current Chatter status.
  - ◊ `created_date:xsd datetime` value of the creation date of the last post by the user, for example, 2010-05-08T05:17:51.000Z
  - ◊ `body`: the body of the post
- `photos`—A map of URLs to the user's profile pictures

  Note: Accessing these URLs requires passing an access token. See "Using the Access Token" in the online help.

  - ◊ `picture`
  - ◊ `thumbnail`
- `urls`—A map containing various API endpoints that can be used with the specified user.

  Note: Accessing the REST endpoints requires passing an access token. See "Using the Access Token" in the online help.

  - ◊ `enterprise` (SOAP)
  - ◊ `metadata` (SOAP)
  - ◊ `partner` (SOAP)
  - ◊ `profile`
  - ◊ `feeds` (Chatter)
  - ◊ `feed-items` (Chatter)
  - ◊ `groups` (Chatter)
  - ◊ `users` (Chatter)
  - ◊ `custom_domain`—This value is omitted if the organization doesn't have a custom domain configured and propagated
- `active`—A boolean specifying whether the queried user is active
- `user_type`—The type of the queried user
- `language`—The queried user's language
- `locale`—The queried user's locale
- `utcOffset`—The offset from UTC of the timezone of the queried user, in milliseconds
- `last_modified_date`—`xsd datetime` format of last modification of the user, for example, 2010-06-28T20:54:09.000Z

The following is a response in XML format:

```
<?xml version="1.0" encoding="UTF-8"?>
<user xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<id>http://na1.salesforce.com/id/00Dx0000001T0zk/005x0000001S2b9</id>
<asserted_user>true</asserted_user>
<user_id>005x0000001S2b9</user_id>
<organization_id>00Dx0000001T0zk</organization_id>
<nick_name>admin1.2777578168398293E12foofoofoofoo</nick_name>
<display_name>Alan Van</display_name>
<email>admin@2060747062579699.com</email>
<status>
   <created_date xsi:nil="true"/>
   <body xsi:nil="true"/>
```

```
</status>
<photos>
   <picture>http://na1.salesforce.com/profilephoto/005/F</picture>
   <thumbnail>http://na1.salesforce.com/profilephoto/005/T</thumbnail>
</photos>
<urls>
   <enterprise>http://na1.salesforce.com/services/Soap/c/{version}/00Dx0000001T0zk
   </enterprise>
   <metadata>http://na1.salesforce.com/services/Soap/m/{version}/00Dx0000001T0zk
   </metadata>
   <partner>http://na1.salesforce.com/services/Soap/u/{version}/00Dx0000001T0zk
   </partner>
   <rest>http://na1.salesforce.com/services/data/v{version}/
   </rest>
   <sobjects>http://na1.salesforce.com/services/data/v{version}/sobjects/
   </sobjects>
   <search>http://na1.salesforce.com/services/data/v{version}/search/
   </search>
   <query>http://na1.salesforce.com/services/data/v{version}/query/
   </query>
   <profile>http://na1.salesforce.com/005x0000001S2b9
   </profile>
</urls>
<active>true</active>
<user_type>STANDARD</user_type>
<language>en_US</language>
<locale>en_US</locale>
<utcOffset>-28800000</utcOffset>
<last_modified_date>2010-06-28T20:54:09.000Z</last_modified_date>
</user>
```

The following is a response in JSON format:

```
{"id":"http://na1.salesforce.com/id/00Dx0000001T0zk/005x0000001S2b9",
"asserted_user":true,
"user_id":"005x0000001S2b9",
"organization_id":"00Dx0000001T0zk",
"nick_name":"admin1.2777578168398293E12foofoofoofoo",
"display_name":"Alan Van",
"email":"admin@2060747062579699.com",
"status":{"created_date":null,"body":null},
"photos":{"picture":"http://na1.salesforce.com/profilephoto/005/F",
   "thumbnail":"http://na1.salesforce.com/profilephoto/005/T"},
"urls":
   {"enterprise":"http://na1.salesforce.com/services/Soap/c/{version}/00Dx0000001T0zk",
   "metadata":"http://na1.salesforce.com/services/Soap/m/{version}/00Dx0000001T0zk",
   "partner":"http://na1.salesforce.com/services/Soap/u/{version}/00Dx0000001T0zk",
   "rest":"http://na1.salesforce.com/services/data/v{version}/",
   "sobjects":"http://na1.salesforce.com/services/data/v{version}/sobjects/",
   "search":"http://na1.salesforce.com/services/data/v{version}/search/",
   "query":"http://na1.salesforce.com/services/data/v{version}/query/",
   "profile":"http://na1.salesforce.com/005x0000001S2b9"},
"active":true,
"user_type":"STANDARD",
"language":"en_US",
"locale":"en_US",
"utcOffset":-28800000,
"last_modified_date":"2010-06-28T20:54:09.000+0000"}
```

After making an invalid request, the following are possible responses from Salesforce:

Request Problem	Error Code
HTTP	403 (forbidden) — HTTPS_Required

Request Problem	Error Code
Missing access token	403 (forbidden) — Missing_OAuth_Token
Invalid access token	403 (forbidden) — Bad_OAuth_Token
Users in a different organization	403 (forbidden) — Wrong_Org
Invalid or bad user or organization ID	404 (not found) — Bad_Id
Deactivated user or inactive organization	404 (not found) — Inactive
User lacks proper access to organization or information	404 (not found) — No_Access
Request to the endpoint of a site	404 (not found) — No_Site_Endpoint
Invalid version	406 (not acceptable) — Invalid_Version
Invalid callback	406 (not acceptable) — Invalid_Callback

Setting a Custom Login Server

For special cases--for example, if you're a Salesforce partner using Trialforce--you might need to redirect your customer login requests to a non-standard login URI. For iOS apps, you set the Custom Host in your app's iOS settings bundle. If you've configured this setting, it will be used as the default connection.

In Android, login hosts are known as server connections. Prior to Mobile SDK v. 1.4, server connections for Android apps were hard-coded in the SalesforceSDK project. In v. 1.4 and later, the host list is defined in the `res/xml/servers.xml` file. The SalesforceSDK library project uses this file to define production and sandbox servers.

You can add your servers to the runtime list by creating your own `res/xml/servers.xml` file in your application project. The root XML element for this file is `<servers>`. This root can contain any number of `<server>` entries. Each `<server>` entry requires two attributes: `name` (an arbitrary human-friendly label) and `url` (the web address of the login server.)

Here's an example of a servers.xml file.

```
<?xml version="1.0" encoding="utf-8"?>
<servers>
  <server name="XYZ.com Login" url="https://<username>.cloudforce.com"/>
</servers>
```

Server Whitelisting Errors

If you get a whitelist rejection error, you'll need to add your custom login domain to the `ExternalHosts` list for your project. This list is defined in the `<project_name>/<platform_path>/config.xml` file. Add those domains (e.g. cloudforce.com) to the app's whitelist in the following files:

For Mobile SDK v. 1.5:

- **iOS:** `/Supporting Files/config.xml`
- **Android:** `/res/xml/config.xml`

For Mobile SDK v. 1.4:

- **iOS:** `/Supporting Files/Cordova.plist`
- **Android:** `/res/xml/Cordova.xml`

For Mobile SDK v. 1.3:

- **iOS:** `/Supporting Files/Cordova.plist`
- **Android:** `/res/xml/cordova.xml`

For Mobile SDK versions prior to v. 1.3:

- **iOS:** `/Supporting Files/PhoneGap.plist`
- **Android:** `/res/xml/PhoneGap.xml`

Revoking OAuth Tokens

When a user logs out of an app, or the app times out or in other ways becomes invalid, the logged-in users' credentials are cleared from the mobile app. This effectively ends the connection to the server, but you can also explicitly revoke the token on the server.

When users request their data from within the external application (the consumer's page), they are authenticated. You can revoke their access tokens, or the refresh token and all related access tokens, using revocation. Developers can use this feature when configuring a Log Out button in their application.

Revoking Tokens

To revoke OAuth 2.0 tokens, use the revocation endpoint:

```
https://login.salesforce.com/services/oauth2/revoke
```

Construct a POST request that includes the following parameters using the `application/x-www-form-urlencoded` format in the HTTP request entity-body. For example:

```
    POST /revoke HTTP/1.1
    Host: https://login.salesforce.com/services/oauth2/revoke
    Content-Type: application/x-www-form-urlencoded

    token=currenttoken
```

If an access token is included, we invalidate it and revoke the token. If a refresh token is included, we revoke it as well as any associated access tokens.

The authorization server indicates successful processing of the request by returning an HTTP status code 200. For all error conditions, a status code 400 is used along with one of the following error responses.

- `unsupported_token_type`—token type not supported
- `invalid_token`—the token was invalid

For Sandbox, use `test.salesforce.com` instead of `login.salesforce.com`.

Handling Refresh Token Revocation in Android Native Apps

Beginning with Salesforce Mobile SDK version 1.5, native Android apps can control what happens when a refresh token is revoked by an administrator. The default behavior in this case is to automatically log out the current user. As a result of this behavior:

- Any subsequent REST API calls your app makes will fail.
- The system discards your user's account information and cached offline data.
- The system forces the user to navigate away from your page.

- The user must log in to Salesforce again to continue using your app.

These side effects provide a secure response to the administrator's action, but they might or might not be suitable for your application. In your code you can choose whether to accept the default behavior or implement your own response. In either case, read the following sections to see if you need to adapt your code:

- Token Revocation Events on page 18
- Token Revocation: Active Handling on page 19
- Token Revocation: Passive Handling on page 18

Token Revocation Events

When a token revocation event occurs, the `ClientManager` object sends an Android-style notification. The intent action for this notification is declared in the `ClientManager.ACCESS_TOKEN_REVOKE_INTENT` constant. `TokenRevocationReceiver`, a utility class, is designed to respond to this intent action. To provide your own handler, you'll extend this class and override the `onReceive()` method. See Token Revocation: Active Handling on page 19.

`NativeMainActivity.java` and `SalesforceDroidGapActivity.java` implement `ACCESS_TOKEN_REVOKE_INTENT` event listeners. These listeners automatically take logged out users to the login page when the refresh token is revoked. A toast message notifies the user of this occurrence.

Token Revocation: Passive Handling

You can let the SDK handle all token revocation events with no active involvement on your part. However, even if you take this passive approach, you might still need to change your code. You do not need to change your code if:

- Your app contains any services, or
- All of your activities extend `NativeMainActivity`.

If your app fails to satisfy at least one of these conditions, implement the following code changes.

1. For legacy apps written before the Mobile SDK 1.5 release: In the `ClientManager` constructor, set the `revokedTokenShouldLogout` parameter to true.

   Note: This step is not necessary for apps that are new in Mobile SDK 1.5 or later.

2. In any activity that does not extend `com.salesforce.androidsdk.ui.NativeMainActivity`, amend the code as follows.

   a. Declare a new variable:

   ```
   private TokenRevocationReceiver tokenRevocationReceiver;
   ```

   b. In the `onCreate()` method add the following code:

   ```
   tokenRevocationReceiver = new TokenRevocationReceiver(this);
   ```

   c. In the `onResume()` method add the following code:

   ```
   registerReceiver(tokenRevocationReceiver, new
   IntentFilter(ClientManager.ACCESS_TOKEN_REVOKE_INTENT));
   ```

d. In the `onPause()` method add the following code:

```
unregisterReceiver(tokenRevocationReceiver);
```

Token Revocation: Active Handling

If you choose to implement your own token revocation event handler, be sure to fully analyze the security implications of your customized flow, and then test it thoroughly. Be especially careful with how you dispose of cached user data. Because the user's access has been revoked, that user should no longer have access to sensitive data.

To provide custom handling of token revocation events:

1. The starting point for implementing your own response is the `ForceApp.shouldLogoutWhenTokenRevoked()` method. By default, this method returns true. Override this method to return false in your `ForceApp` subclass.

```
@Override
public boolean shouldLogoutWhenTokenRevoked() {
 return false;
}
```

2. The `ClientManager` constructor provides a boolean parameter, `revokedTokenShouldLogout`. Set this parameter to false. You can do this by calling `shouldLogoutWhenTokenRevoked()` on your `ForceApp` subclass.
3. Implement your handler by extending `TokenRevocationReceiver` and overriding the `onReceive()` method.
4. Regardless of whether your activity subclasses `NativeMainActivity`, perform step 2 in Token Revocation: Passive Handling on page 18.

Connected Apps

A Connected App is an application that integrates with salesforce.com using Identity and Data APIs, such as SOAP API or REST API. Connected Apps use the standard OAuth 2.0 protocol to authenticate, provide Single Sign-On, and provide tokens for use with Salesforce APIs. In addition to standard OAuth capabilities, Connected Apps allow Administrators explicit control over who may use the application, and various security policies which should be enforced.

Connected Apps begin with a developer defining OAuth metadata about the application, including:

- Basic descriptive and contact information for the Connected App
- The OAuth scopes and callback URL for the Connected App
- Optional IP ranges where the Connected App might be running
- Optional information about mobile policies the Connected App can enforce

In return, the developer is provided an OAuth client Id and client secret, as well as an install URL for the Connected App. The developer can then provide this URL to a Salesforce administrator.

The administrator can install the Connected App into their organization and use profiles, permission sets, and IP range restrictions to control which users can access the application. Management is done from a detail page for the Connected App. The administrator can also uninstall the Connected App and install a newer version. When the app is updated, the developer can notify administrators that there is a new version available for the app—their existing installation URL installs the new version.

About PIN Security

Mobile Connected Apps have an additional layer of security via PIN protection on the app. This PIN protection is for the mobile app itself, and isn't the same as the PIN protection on the device or the login security provided by the Salesforce organization.

In order to use PIN protection, the developer must select the **Implements Screen Locking & Pin Protection** checkbox when creating the Connected App. Mobile app administrators then have the option of enforcing PIN protection and customizing timeout duration and PIN length.

Note: Because PIN security is implemented in the mobile device's operating system, only native and hybrid mobile apps can use PIN protection; HTML5 Web apps can't use PIN protection.

In practice, PIN protection can be used so that the mobile app locks up if it's isn't used for a specified number of minutes. Note that when a mobile app is sent to the background, the clock continues to tick.

To illustrate how PIN protection works:

1. User turns on phone and enters PIN for the device.
2. User starts mobile app (Connected App).
3. User enters login information for Salesforce organization.
4. User enters PIN code for mobile app.
5. User works in the app, and then sends it to the background by opening another app (or receiving a call, etc).
6. The mobile app times out.
7. User re-opens the app, and the app PIN screen displays (for the mobile app, not the device).
8. User enters app PIN, and can resume working.

Chapter 3

Native iOS Development

In this chapter ...

This topic introduces the chapter on native iOS development.

The two main things the iOS native SDK provides:

- Automation of the OAuth2 login process, making it easy to integrate OAuth with your app.
- Access to the REST API with all the infrastructure classes (including third-party libraries such as RestKit) for making that access as easy as possible.

When you create a new project using the Salesforce Mobile SDK, a template application is included automatically. This simple app allows you to connect to a Salesforce organization and run a simple query. It doesn't do much, but it lets you know things are working as designed.

iOS Native Quick Start

Use the following procedure to get started quickly.

1. Make sure you meet all of the native iOS requirements.
2. Install the Mobile SDK for iOS.
3. Run the template app.

Native iOS Requirements

- XCode—4.0 is the minimum, but we recommend the latest version.
- iOS 5.0 or greater
- Install the Mobile SDK.
- A Developer Edition organization with a remote access application.

For important information on using various versions of XCode, see the Readme at https://github.com/forcedotcom/SalesforceMobileSDK-iOS/blob/master/readme.md.

Tips for using Xcode versions earlier than 4.5 (iOS 6)

The iOS 6 development environment introduces a new processor architecture (armv7s), while removing support for another (armv6). This makes libraries built for iOS 6 incompatible with earlier Xcode dev environments, without some further steps on the developer's part.

We have updated our libraries to support iOS 6. If you're not ready to update your Xcode environment to 4.5, you'll need to remove support for ARM v. 6 if it exists in the SalesforceHybridSDK project (`hybrid/SalesforceHybridSDK/SalesforceHybridSDK.xcodeproj`), as well as in your app's project configuration.

1. In Xcode, click on the Project at the top of the Project Navigator view.
2. Under the Project header, click the project/app name.
3. Select the Build Settings tab.
4. Review the Architectures section at the top. If **armv6** is not listed in Valid Architectures, no action is needed. Otherwise, double-click the **armv6 armv7** option in Valid Architectures.
5. Click on the **armv6** item in the list, then click **—** at the bottom of the list.

These steps make your project compatible with the iOS 6 library builds. You lose nothing by removing support for ARM v. 6 devices, which were never compatible with the iOS 5.0 SDK version (the baseline version supported by the Mobile SDK.)

Note:

Xcode 4.3 Users: If you have not used the `xcode-select` tool to choose your version of Xcode at the command line, you might encounter the following error when running the install script:

```
xcode-select: Error: No Xcode folder is set. Run xcode-select -switch
<xcode_folder_path> to set the path to the Xcode folder.
```

To run the install script without issues, use the following command:

```
sudo xcode-select -switch /Applications/Xcode.app/Contents/Developer
```

This example assumes you installed Xcode in the Applications folder.

Installing the Mobile SDK for iOS

1. In your browser, navigate to the Mobile SDK iOS GitHub repository: `https://github.com/forcedotcom/SalesforceMobileSDK-iOS`.
2. Clone the repository to your local file system by issuing the following command: `git clone git://github.com/forcedotcom/SalesforceMobileSDK-iOS.git`

   Note: If you have the GitHub app for Mac OS X, click **Clone in Mac**.

3. Open the OS X Terminal app in the directory where you installed the cloned repository and run the install script from the command line: `./install.sh`
4. You also need to download the sample app from GitHub: `https://github.com/forcedotcom/SalesforceMobileSDK-Samples/tree/master/iOS/CloudTunesNative`.

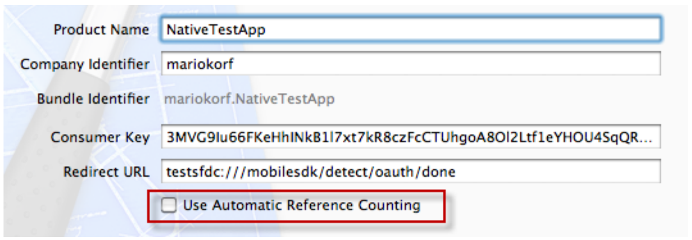
Creating a New Native iOS App in Xcode

Use the following procedure to create and configure a new Force.com–based Application project.

1. Open Xcode and create a new project (Shift-Command-N).
2. Select **Native Force.com REST App** and click **Next**.
3. In the `Choose options for your new project` dialog, enter `NativeTestApp`.

   Note: You might also need to enter a Company Identifier prefix if you haven't used Xcode before.

4. Make sure the checkbox for **Use Automatic Reference Counting** is cleared.



Product Name	NativeTestApp
Company Identifier	mariokorf
Bundle Identifier	mariokorf.NativeTestApp
Consumer Key	3MVG9Iu66FKeHhINkB1l7xt7kR8czFcCTUhgoA8Ol2Ltf1eYHOU4SqQR...
Redirect URL	testsfdc:///mobilesdk/detect/oauth/done

☐ Use Automatic Reference Counting

5. Click **Next**.
6. Specify a location for your new project and click **Create**.

Running the Xcode Project Template App

The Xcode project template includes a sample application you can run right away.

1. Press **Command-R** and the default template app runs in the iOS simulator.

   Note: If you get build errors, make sure **Automatic Reference Counting (ARC)** is turned off.

   a. Select your project in the Navigator.
   b. In the Build Settings tab, toggle the `Objective-C Automatic Reference Counting` value to `No`.

2. On startup, the application starts the OAuth authentication flow, which results in an authentication page. Enter your credentials, and click **Login**.
3. Tap **Allow** when asked for permission

You should now be able to compile and run the sample project. It's a simple app that logs you into an org via OAuth2, issues a `'select Name from Account'` query, and displays the result in a `UITableView`.

Using the Mobile SDK in an Existing Project

If you want to incorporate the Mobile SDK into an existing iOS project, do the following.

1. In Xcode, drag the folder `native/dependencies` into your project (select **Create groups** for any added folders).
2. Open the **Build Settings** tab for the project and set `Other Linker Flags` to `-ObjC -all_load`.
3. Open the **Build Phases** tab for the project main target and link against the following required frameworks:
   - `CFNetwork.framework`
   - `CoreData.framework`
   - `MobileCoreServices.framework`
   - `SystemConfiguration.framework`
   - `Security.framework`
   - `libxml2.dylib`
4. Import the `SalesforceSDK` header via `#import "SFRestAPI.h"`.
5. Build the project to verify that the installation is successful.
6. Refer to the SFRestAPI documentation for some sample code to log into a Salesforce instance and issue a REST API call.

iOS RestAPIExplorer Sample Application

The Running the Xcode Project Template App is itself a sample application, but it only does one thing: issue a SOQL query and return a result. The `RestAPIExplorer` sample app has a lot more functionality you can examine and work into your own apps.

The RestAPIExplorer sample app is in the Mobile SDK for iOS under `native/SampleApps/RestAPIExplorer`.

Chapter 4

Native Android Development

In this chapter ...

This topic introduces the chapter on native Android development.

The two main things the Android native SDK provides are:

- Automation of the OAuth2 login process, making it easy to integrate the process with your app.
- Access to the REST API, with infrastructure classes that simplify that access.

The Android Salesforce Mobile SDK includes several sample native applications. We also provide an ant target to quickly create a new application.

Android Native Quick Start

Use the following procedure to get started quickly.

1. Make sure you meet all of the native Android requirements.
2. Install the Mobile SDK for Android.
3. At the command line, run an ant script to create a new Android project , and then run the template application from the command line.
4. Set up your projects in Eclipse.

Native Android Requirements

- Java JDK 6.
- Apache Ant 1.8 or later.
- Android SDK, version 21 or later—`http://developer.android.com/sdk/installing.html`.

  Note: For best results, install all previous versions of the Android SDK as well as your target version.

- Eclipse 3.6 or later. See `http://developer.android.com/sdk/requirements.html` for other versions.
- Android ADT (Android Development Tools) plugin for Eclipse, version 21 or later—`http://developer.android.com/sdk/eclipse-adt.html#installing`.
- In order to run the application in the Emulator, you need to set up at least one Android Virtual Device (AVD) that targets Platform 2.2 or above (we recommend 2.2). To learn how to set up an AVD in Eclipse, follow the instructions at `http://developer.android.com/guide/developing/devices/managing-avds.html`.
- A Developer Edition organization with a remote access application.

The `SalesforceSDK` project is built with the Android 3.0 (Honeycomb) library. The primary reason for this is that we want to be able to make a conditional check at runtime for file system encryption capabilities. This check is bypassed on earlier Android platforms; thus, you can still use the `salesforcesdk.jar` in earlier Android application versions, down to the mininum-supported Android 2.2.

Installing the Mobile SDK for Android

1. In your browser, navigate to the Mobile SDK Android GitHub repository: https://github.com/forcedotcom/SalesforceMobileSDK-Android.
2. Clone the repository to your local file system by issuing the following command: `git clone git://github.com/forcedotcom/SalesforceMobileSDK-Android.git`
3. Open a command prompt in the directory where you installed the cloned repository, and run the install script from the command line: `./install.sh`

Note: Windows users run `cscript install.vbs`.

Create shell variables:

1. `ANDROID_SDK_DIR` pointing to the Android SDK directory
2. `SALESFORCE_SDK_DIR` pointing to your clone of the Salesforce Mobile SDK repository, for example: `/home/jon/SalesforceMobileSDK-Android`
3. `NATIVE_DIR` pointing to `$SALESFORCE_SDK_DIR/native`
4. `TARGET_DIR` pointing to a location you've defined to contain your Android project

Note: If you haven't set up these variables, make sure to replace `$ANDROID_SDK_DIR`, `$SALESFORCE_SDK_DIR`, `$NATIVE_DIR` and `$TARGET_DIR` in the various code snippets in this guide with the actual paths.

Creating a New Android Project

We've made it easy to create a native Android project by using an ant script.

1. Open a command prompt in the location where you installed the SDK (or `$SALESFORCE_SDK_DIR` if you created that variable).
2. Enter `ant create_native -Dapp.name={appName} -Dtarget.dir={targetDir} -Dpackage.name={packageName} [-Duse.smartstore=true]`

   Where:

   - `appName` is the name of the new application.
   - `targetDir` is the directory where the code should reside.
   - `packageName` is the Java package for the new application (for example, `com.acme.mobileapp`.)
   - `use.smartstore=true` is an optional flag for including the SmartStore library in your application. If you don't intend to use SmartStore for offline storage, you can keep your app lightweight by omitting this parameter or setting it to false.

If this is your first time building an application with the Salesforce Mobile SDK, also enter the following:

```
cd $NATIVE_DIR/SalesforceSDK
$ANDROID_SDK_DIR/android update project -p .
```

The Android project you created contains a simple application you can build and run.

Android Template Application

The template native app for Android allows you to login and do standard CRM tasks, such as queries and inserts.

To build the new application, do the following:

1. In a text editor, open `$TARGET_DIR/res/values/rest.xml`.

2. Enter your OAuth client ID and callback URL, and then save the file.
3. Open a command prompt and enter the following commands:

```
cd $TARGET_DIR
$ANDROID_SDK_DIR/tools/android update project -p . -t 1
ant clean debug
```

Note: The `-t <id>` parameter specifies the target Android version. Use `android.bat list targets` to see the IDs for API versions installed on your system.

4. If your emulator is not running, use the Android AVD Manager to start it. If you are using a real device, connect it.
5. Enter `ant installd`.

Setting Up Projects in Eclipse

The repository you cloned has other sample apps you can run. To import those into Eclipse:

1. Launch Eclipse and select `$TARGET_DIR` as your workspace directory.
2. Select **Window** > **Preferences**, choose the **Android** section, and enter the the Android SDK location.
3. Click OK.
4. Select **File** > **Import** and select **General** > **Existing Projects into Workspace**.
5. Click Next.
6. Select `$NATIVE_DIR` as your root directory and import the projects listed in Android Project Files.

If you are using an older version of the Mobile SDK, you may need to create some project folders in the sample projects.

1. In the navigator, right-click SalesforceSDKTest, choose **New** > **Folder**, and set the folder name to `gen`. If the folder already exists, click Cancel.
2. In the same way, create a `gen` folder for `RestExplorer` and `RestExplorerTest` projects.
3. Right–click the `SalesforceSDK` project and create a folder called `res`.

Android Project Files

Inside the $NATIVE_DIR, you will find several projects:

1. `SalesforceSDK`—The SalesforceSDK, which provides support for OAuth2 and REST API calls
2. `test/SalesforceSDKTest`—Tests for the SalesforceSDK project
3. `TemplateApp`—Template used when creating new native applications using SalesforceSDK
4. `test/TemplateAppTest`—Tests for the TemplateApp project
5. `SampleApps/RestExplorer`—App using SalesforceSDK to explore the REST API calls
6. `SampleApps/test/RestExplorerTest` —Tests for the RestExplorer project
7. `SampleApps/CloudTunes`—A sample native application using SalesforceSDK, used in the Mobile SDK Workbook

Android RestExplorer Sample Application

The RestExplorer is a sample app that demonstrates how to use the OAuth and REST API functions of the SalesforceSDK. It is also useful to investigate the various REST API actions from a Honeycomb tablet.

1. To run the application, right-click the **RestExplorer** project and choose **Run As** > **Android Application**.
2. To run the tests, right-click the **RestExplorerTest** project and choose **Run As** > **Android JUnit Test**.

Chapter 5

Introduction to Hybrid Development

In this chapter ...

Hybrid apps combine the ease of HTML5 Web app development with the power and features of the native platform.

Hybrid apps depend on HTML and JavaScript files. These files can be stored on the device or on the server.

- **Device** — Hybrid apps developed with `forcetk` wrap a Web app inside the Salesforce Mobile Container. In this methodology, the JavaScript and HTML files are stored on the device.
- **Salesforce Mobile Container** — Hybrid apps developed using Visualforce technology store the HTML and JavaScript files on the server and are delivered through the native container.

iOS Hybrid Development

In order to develop hybrid applications, you'll need to meet some of the prerequisites for both the iOS native and the vanilla HTML5 scenarios.

1. Make sure you meet the HTML5 Development Requirements
2. Follow the installation instructions for iOS.

Note:

SECURITY WARNING: Hybrid iOS apps run in a web view container that caches HTTP response data. Beginning in iOS 5.1, all HTTP responses are cached by default in a file system database in plain text. The database file path, as of this writing, is `<Application>/Library/Cache/<application name>/Cache.db`. Although this cache system enhances performance, it could also expose sensitive data to security risks. We recommend that you disable the cache unless you have good reasons for using it.

To disable caching, add the following header to your HTTP requests:

```
Cache-Control: no-cache, no-store
```

iOS Hybrid Sample Application

The sample applications contained under the `hybrid/SampleApps` folder are designed around the PhoneGap SDK. PhoneGap is also known as Cordova. Salesforce Mobile SDK v. 1.4 and later include the Cordova libraries, so no separate installation is required. You can find documentation for the Cordova SDK in the Getting Started Guide.

Inside the `hybrid/SampleApps` foler, you can find sample projects:

- **ContactExplorer**: The `ContactExplorer` sample app uses PhoneGap (also known as Cordova) to retrieve local device contacts. It also uses the `forcetk.js` toolkit to implement REST transactions with the Salesforce REST API. The app uses the OAuth2 support in Salesforce SDK to obtain OAuth credentials, then propagates those credentials to `forcetk.js` by sending a javascript event.
- **SFDCAccount**: A simple hybrid application that demonstrates how to take accounts and opportunities offline using SmartStore and forcetk.
- **VFConnector**: The VFConnector sample app demonstrates how to wrap a Visualforce page in a native container. This example assumes that your org has a Visualforce page called `BasicVFTest`. The app first obtains OAuth login credentials using the Salesforce SDK OAuth2 support, then uses those credentials to set appropriate webview cookies for accessing Visualforce pages.
- **SmartStoreExplorer**: Lets you explore SmartStore APIs.

Android Hybrid Development

In order to develop hybrid applications, you'll need to meet some of the prerequisites for both the Android native and the vanilla HTML5 scenarios.

1. Make sure you meet the HTML5 Development Requirements.

2. Follow the installation instructions for Android Native.
3. Create an additional shell variable called `HYBRID_DIR` that points to `$SALESFORCE_SDK_DIR/hybrid`.
4. Run the sample app from the command line.

There is a `build.xml` file in `$SALESFORCE_SDK_DIR`. If you simply type: `ant` the build file prints information about available targets.

To create a new hybrid application with local HTML and JavaScript files, use the `create_hybrid_local` target:

```
ant create_hybrid_local -Dapp.name={appName} -Dtarget.dir={targetDir}
-Dpackage.name={packageName} [-Duse.smartstore=true]
```

Where:

- `appName` is the name for the new application.
- `targetDir` is the directory where the code should reside.
- `packageName` is the Java package for the new application (for example, `com.acme.mobileapp`.)
- `use.smartstore=true` is an optional flag for including the SmartStore library in your application. If you don't intend to use SmartStore for offline storage, you can keep your app lightweight by omitting this parameter or setting it to false.

Put your HTML and JavaScript files in `${target.dir}/assets/www/`.

To create a new hybrid application with remote HTML and JavaScript, use the `create_hybrid_vf` target::

```
ant create_hybrid_vf -Dapp.name={appName} -Dtarget.dir={targetDir}
-Dpackage.name={packageName} -Dapex.page={apexPage} [-Duse.smartstore=true]
```

Where:

- `appName` is the name of the new application.
- `targetDir` is the directory where the code should reside.
- `packageName` is the Java package for the new application (for example, `com.acme.mobileapp`.)
- `apexPage` is the Apex page for the application (for example, `/apex/MyFirstApp`)
- `use.smartstore=true` is an optional flag for including the SmartStore library in your application. If you don't intend to use SmartStore for offline storage, you can keep your app lightweight by omitting this parameter or setting it to false.

Android Hybrid Sample Applications

Inside the `HYBRID_DIR`, you can find sample projects and related test applications:

- **SampleApps/ContactExplorer**: The `ContactExplorer` sample app uses PhoneGap (also known as Cordova) to retrieve local device contacts. It also uses the `forcetk.js` toolkit to implement REST transactions with the Salesforce REST API. The app uses the OAuth2 support in Salesforce SDK to obtain OAuth credentials, then propagates those credentials to `forcetk.js` by sending a javascript event.
- **SampleApps/test/ContactExplorerTest**: Tests for the ContactExplorer sample app.
- **SampleApps/SFDCAccount**: A simple hybrid application that demonstrates how to take accounts and opportunities offline using SmartStore and forcetk.
- **SampleApps/test/SFDCAccountTest**: Tests for the SFDCAccount sample app.

- **SampleApps/VFConnector**: The VFConnector sample app demonstrates how to wrap a Visualforce page in a native container. This example assumes that your org has a Visualforce page called `BasicVFTest`. The app first obtains OAuth login credentials using the Salesforce SDK OAuth2 support, then uses those credentials to set appropriate webview cookies for accessing Visualforce pages.
- **SampleApps/test/VFConnectorTest**: Test for the VFConnector sample app.
- **SampleApps/SmartStoreExplorer**: Lets you explore SmartStore APIs.
- **SampleApps/test/SmartStoreExplorerTest**: Tests for the SmartStoreExplorer sample app.

Visualforce Mobile Open Components

The flexible component infrastructure of Visualforce makes it possible to wrap low-level code into reusable components for developing custom apps on the Force.com platform. You can leverage Visualforce to create hybrid mobile applications.

The open-source Mobile Components for Visualforce library contains lightweight UI components that generate cross-platform HTML5 output. These apps can be deployed in the browser or embedded inside the hybrid Mobile SDK container. The source code for this library is available on Github: https://github.com/forcedotcom/MobileComponents/tree/master/Visualforce

The component library enables any Visualforce developer to quickly and easily build mobile applications without having to dig deep into complex mobile frameworks and design patterns. This library includes the frameworks and best practices of mobile development that can be used with simple component interfaces.

The Visualforce library consists of the following components:

Note: The following open-source Visualforce components are not officially supported by Salesforce.

- **App Component** — This component provides all the settings and architecture pieces (including jQuery and jQuery Mobile) for mobile app development.
- **SplitView Template** — This template provides the split view components on the page. In landscape mode, it provides a left menu section and a broad main section. In portrait mode, it turns the left menu into a popover.
- **List Component** — List Component provides a quick and easy way to render a record list for any sObject. One can easily manage the behavior of the component by using the various attributes or the JavaScript hooks on this component.
- **Detail Component** — Detail Component provides a quick and easy way to render the details for any sObject. One can easily manage the behavior by using the various attributes or the JavaScript hooks.
- **Page Component** — Page Component provides a jQuery Mobile wrapper with `data-role="page"`.
- **Header Component** — Header Component provides a jQuery Mobile wrapper with `data-role="header"` for header sections inside a Page component.
- **Content Component** — Content Component provides a jQuery Mobile wrapper with `data-role="content"` for content sections inside a Page component.
- **Footer Component** — Footer Component provides a jQuery Mobile wrapper with

  ```
  data-role="footer"
  ```

  for footer sections inside a Page component.
- **Navigation Component** — Navigation Component can be used to create hooks for navigation between various jQuery Mobile pages.

Visualforce App Component

All mobile Visualforce pages built using this component library need to be wrapped inside the App component. The App component provides the primary architectural pieces, such as viewport settings, javascripts, stylesheets etc., for the mobile app. The `debug` attribute on App components lets you specify if you are running in development or production mode, and delivers the minified version of assets for the latter case.

```
 <c:App debug="true"></c:App>
```

Visualforce SplitView Template

The SplitViewTemplate is used inside the App component. The split view page provides a slim left section for list view, and a wider right section to show the record details.

```
<apex:composition template="SplitViewTemplate"></apex:composition>
```

SplitViewTemplate consists of two sections you need to define.

- "`menu`" is the left section of the split view in landscape mode. This section becomes a popover when a user rotates the tablet to switch to the portrait mode.
- "`main`" is the right wide section of the split view. This section is always visible on the tablet in both portrait and landscape modes.

```
<apex:define name="menu">
     <c:Page name="list">
         <c:Header >
             <h1 style="font-size: 20px; margin: 0px;">All Contacts</h1>
         </c:Header>
         <c:Content >
             <c:List sobject="Contact" labelField="Name" subLabelField="Account.Name"
sortByField="FirstName" listFilter="true"/>
         </c:Content>
     </c:Page>
 </apex:define>
```

Visualforce Page Component

The Page component provides an important wrapper to define the fixed header and footer components with a scrollable content section in between.

- The Header component is used to define the fixed title of the section.
- The Content component describes the scrollable content section.
- Within the body of Content component, the List component is used to fetch and display the contact list.

Visualforce Header Component

Provides a jQuery Mobile wrapper with `data-role="header"` for header sections inside a Page component. Header components often include a <H1> tag. The header and footer components are fixed, and scrollable content can go between them.

Visualforce Content Component

The Content component describes the scrollable content section, including the List and Detail components.

Within the Content component, the List and Detail components respect the Object, Field, and Record visibility settings so you can develop the applications using these components without worrying about data security. Also, one can easily override the CSS styles of these components to give it a look and feel as required by your project. If you have used jQuery mobile, you can easily see that the page, header, footer, and content components actually use the jQuery mobile properties to enable the mobile user experience. So, while using this component library, you can easily leverage other features of jQuery mobile too.

Visualforce List Component

The List component is used to fetch and display records. The sObject attribute on List component is used to specify the sObject type for which the list view needs to be created. Other attributes, such as labelField, subLabelField, sortByField, and listFilter, are used to specify the behavior and display properties of this list.

```
<c:List sobject="Contact" labelField="Name" subLabelField="Account.Name"
sortByField="FirstName" listFilter="true"/>
```

Visualforce Detail Component

The Detail component takes the sObject type as an attribute and generates the mobile layout with details of the associated record. The mobile layout of the record is driven by the Page layout associated to the current user profile and associated record type. This gives the administrator the flexibility to update the mobile layout by using the standard Salesforce page layout manager without further code modifications.

```
<apex:define name="main">
     <c:Page name="main">
         <c:Header >
             <h1 style="font-size: 22px; margin: 0px;">Contact Details</h1>
         </c:Header>
         <c:Content >
             <c:Detail sobject="Contact"/>
         </c:Content>
     </c:Page>
</apex:define>
```

Visualforce Footer Component

Provides a jQuery Mobile wrapper with `data-role="footer"` for header sections inside a Page component. The header and footer components are fixed, and scrollable content can go between them.

Visualforce Navigation Component

The navigation component provides a way to create hooks for navigation between various jQuery Mobile pages.

Versioning and Javascript Library Compatibility

In hybrid applications, client Javascript code interacts with native code through Cordova (formerly PhoneGap) and SalesforceSDK plugins. When you package your Javascript code with your mobile application, your testing assures that the code works with native code. However, when the Javascript code comes from the server—for example, when the application is written with VisualForce—harmful conflicts can occur. In such cases you must be careful to use Javascript libraries from the version of PhoneGap or Cordova that matches the Mobile SDK version you're using.

For example, suppose you shipped an application with Mobile SDK 1.2, which uses PhoneGap 1.2. Later, you ship an update that uses Mobile SDK 1.3. The 1.3 version of the Mobile SDK uses Cordova 1.8.1 rather than PhoneGap 1.2. You must make sure that the Javascript code in your updated application accesses native components only through the Cordova 1.8.1 and Mobile SDK 1.3 versions of the Javascript libraries. Using mismatched Javascript libraries can crash your application.

You can't force your customers to upgrade their clients, so how can you prevent crashes? First, identify the version of the client. Then, you can either deny access to the application if the client is outdated (for example, with a "Please update to the latest version" warning), or, preferably, serve compatible Javascript libraries.

The following table correlates Cordova and PhoneGap versions to Mobile SDK versions.

Mobile SDK version	Cordova or PhoneGap version
1.2	PhoneGap 1.2
1.3	Cordova 1.8.1
1.4	Cordova 2.2
1.5	Cordova 2.3

Using the User Agent to Find the Mobile SDK Version

Fortunately, you can look up the Mobile SDK version in the user agent. The user agent starts with `SalesforceMobileSDK/<version>`. Once you obtain the user agent, you can parse the returned string to find the Mobile SDK version.

You can obtain the user agent on the server with the following Apex code:

```
userAgent = ApexPages.currentPage().getHeaders().get('User-Agent');
```

On the client, you can do the same in Javascript using the `navigator` object:

```
userAgent = navigator.userAgent;
```

Detecting the Mobile SDK Version with the sdkinfo Plugin

In Javascript, you can also retrieve the Mobile SDK version and other information by using the `sdkinfo` plugin. This plugin, which is defined in the `SFHybridApp.js` file, offers one method:

```
getInfo(callback)
```

This method returns an associative array that provides the following information:

Member name	Description
`sdkVersion`	Version of the Salesforce Mobile SDK used to build to the container. For example: “1.4”.
`appName`	Name of the hybrid application.
`appVersion`	Version of the hybrid application.
`forcePluginsAvailable`	Array containing the names of Salesforce plugins installed in the container. For example: "com.salesforce.oauth", "com.salesforce.smartstore", and so on.

The following code retrieves the information stored in the `sdkinfo` plugin and displays it in alert boxes.

```
var sdkinfo = cordova.require("salesforce/plugin/sdkinfo");
sdkinfo.getInfo(new function(info) {
    alert("sdkVersion->" + info.sdkVersion);
    alert("appName->" + info.appName);
    alert("appVersion->" + info.appVersion);
    alert("forcePluginsAvailable->" + JSON.stringify(info.forcePluginsAvailable));
});
```

See Also:

Example: Serving the Appropriate Javascript Libraries

Managing Sessions in Hybrid Applications

Mobile users expect their apps to just work. To help iron out common difficulties that plague many mobile apps, the Mobile SDK uses native containers for hybrid applications. These containers provide seamless authentication and session management by abstracting the complexity of web session management. However, as popular mobile app architectures evolve, this “one size fits all” approach proves to be too limiting in some cases. For example, if a mobile app uses JavaScript remoting in Visualforce, Salesforce cookies can be lost if the user lets the session expire. These cookies can be retrieved only when the user manually logs back in.

Mobile SDK 1.4 begins to transition hybrid apps away from predefined, proactive session management to more flexible, reactive session management. Rather than letting the hybrid container automatically control the session, developers can participate in the management by responding to session events. This change gives developers more control over managing sessions in the Salesforce Touch Platform.

To switch to reactive management, adjust your session management settings according to your app’s architecture. This table summarizes the behaviors and recommended approaches for common architectures.

App Architecture	Proactive Behavior in SDK 1.3 and Earlier	Reactive Behavior in SDK 1.4	Steps for Upgrading Code
REST API	Background session refresh	Refresh from JavaScript	No change for forcetk.js. For other frameworks, add refresh code.

App Architecture	Proactive Behavior in SDK 1.3 and Earlier	Reactive Behavior in SDK 1.4	Steps for Upgrading Code
JavaScript Remoting in Visualforce	Restart app	Refresh session and CSRF token from JavaScript	Catch timeout, then either reload page or load a new iFrame.
JQuery Mobile	Restart app	Reload page	Catch timeout, then reload page.

These sections provide detailed coding steps for each architecture.

REST APIs (Including Apex2REST)

If you're writing or upgrading a hybrid app that leverages REST APIs, detect an expired session and request a new access token at the time the REST call is made. We encourage authors of apps based on this framework to leverage API wrapping libraries, such as forcetk.js, to manage session retention.

The following code, from `index.html` in the ContactExplorer sample application, demonstrates the recommended technique. When the application first loads, call `getAuthCredentials()` on the Salesforce OAuth plugin, passing the handle to your refresh function (in this case, `salesforceSessionRefreshed`.) The OAuth plugin function calls your refresh function, passing it the session and refresh tokens. Use these returned values to initialize forcetk.

- From the `onDeviceReady()` function:

```
cordova.require("salesforce/plugin/oauth").getAuthCredentials(salesforceSessionRefreshed,
 getAuthCredentialsError);
```

- `salesforceSessionRefreshed()` function:

```
function salesforceSessionRefreshed(credsData) {
        forcetkClient = new forcetk.Client(credsData.clientId, credsData.loginUrl);
        forcetkClient.setSessionToken(credsData.accessToken, apiVersion,
credsData.instanceUrl);
        forcetkClient.setRefreshToken(credsData.refreshToken);
        forcetkClient.setUserAgentString(credsData.userAgent);
    }
```

For the complete code, see the ContactExplorer sample application (`SalesforceMobileSDK-Android\hybrid\SampleApps\ContactExplorer`).

JavaScript Remoting in Visualforce

For mobile apps that use JavaScript remoting to access Visualforce pages, incorporate the session refresh code into the method parameter list. In JavaScript, use the Visualforce remote call to check the session state and adjust accordingly.

```
<Controller>.<Method>(
    <params>,
    function(result, event) {
   if (hasSessionExpired(event)) {
     // Reload will try to redirect to login page, container will intercept
     // the redirect and refresh the session before reloading the origin page
     window.location.reload();
   } else {
     // Everything is OK. You can go ahead and use the result.
  },
    {escape: true}
);
```

This example defines `hasSessionExpired()` as:

```
function hasSessionExpired(event) {
    return (event.type == "exception" && event.message.indexOf("Logged in?") != -1);
}
```

Advanced developers: Reloading the entire page might not provide the optimal user experience. If you want to avoid reloading the entire page, you'll need to:

1. Refresh the access token
2. Refresh the Visualforce domain cookies
3. Finally, refresh the CSRF token

In `hasSessionExpired()`, instead of fully reloading the page as follows:

```
window.location.reload();
```

Do something like this:

```
// Refresh oauth token
cordova.require("salesforce/plugin/oauth").authenticate(
  function(creds) {
   // Reload hidden iframe that points to a blank page to
   // to refresh Visualforce domain cookies
   var iframe = document.getElementById("blankIframeId");
   iframe.src = src;

   // Refresh CSRF cookie
   <provider>.refresh(function() {
     <Retry call for a seamless user experience>;
   });

  },
  function(error) {
    console.log("Refresh failed");
  }
);
```

JQuery Mobile

JQueryMobile makes Ajax calls to transfer data for rendering a page. If a session expires, a 302 error is masked by the framework. To recover, incorporate the following code to force a page refresh.

```
$(document).on('pageloadfailed', function(e, data) {
  console.log('page load failed');
  if (data.xhr.status == 0) {
    // reloading the VF page to initiate authentication
    window.location.reload();
  }
});
```

Example: Serving the Appropriate Javascript Libraries

To provide the correct version of Javascript libraries, create a separate bundle for each Salesforce Mobile SDK version you use. Then, provide Apex code on the server that downloads the required version.

1. For each Salesforce Mobile SDK version that your application supports, do the following.

   a. Create a ZIP file containing the Javascript libraries from the intended SDK version.

   b. Upload the ZIP file to your org as a static resource.

   For example, if you ship a client that uses Salesforce Mobile SDK v. 1.3, add these files to your ZIP file:

   - `SFHybridApp.js`
   - `SalesforceOAuthPlugin.js`
   - `bootconfig.js`
   - `cordova-1.8.1.js`, which you should rename as `cordova.js`

   Note: In your bundle, it's permissible to rename the Cordova Javascript library as `cordova.js` (or `PhoneGap.js` if you're packaging a version that uses a `PhoneGap-x.x.js` library.)

2. Create an Apex controller that determines which bundle to use. In your controller code, parse the user agent string to find which version the client is using.

   a. In your org, go to **Your name** > **Setup** > **App Setup** > **Develop** > **Apex Class**.

   b. Create a new Apex controller named `SDKLibController` with the following definition.

```
public class SDKLibController {
    public String getSDKLib() {
        String userAgent = ApexPages.currentPage().getHeaders().get('User-Agent');

        if (userAgent.contains('SalesforceMobileSDK/1.3')) {
          return 'sdklib13';
        }
// Add additional if statements for other SalesforceSDK versions
// for which you provide library bundles.
    }
}
```

3. Create a Visualforce page for each library in the bundle, and use that page to redirect the client to that library.
   For example, for the SalesforceOAuthPlugin library:

   a. In your org, go to **Your Name** > **Setup** > **App Setup** > **Develop** > **Pages**.

   b. Create a new page called "SalesforceOAuthPlugin" with the following definition.

```
<apex:page controller="SDKLibController" action="{!URLFor($Resource[SDKLib],
 'SalesforceOAuthPlugin.js')}">
</apex:page>
```

   c. Reference the VisualForce page in a `<script>` tag in your HTML code. Be sure to point to the page you created in step 3b. For example:

```
<script type="text/javascript" src="/apex/SalesforceOAuthPlugin" />
```

Note: Provide a separate `<script>` tag for each library in your bundle.

Chapter 6

HTML5 Development

In this chapter ...

HTML5 lets you create lightweight interfaces without installing software on the mobile device; any mobile, touch or desktop device can access the same interface. You can create an HTML5 application that uses Visualforce to deliver the HTML content and fetches record data from Force.com using JavaScript remoting for Apex controllers. The sample application also utilizes the jQuery Mobile library for the user interface.

HTML5 Development Requirements

- You'll need a Force.com organization.

  Note: Since this hybrid development uses Visualforce, you can't use Database.com.

- You must create a remote access application. See Creating a Remote Access Application.
- Some knowledge of Apex and Visualforce is necessary.

Accessing Data Using JavaScript

HTML5 apps require two static resource files, one for JQuery libraries and another for JavaScript remoting.

- The jQuery static resource contains all the JavaScript and stylesheet files for the jQuery and jQuery Mobile libraries.
- You'll also need a JavaScript file containing the methods that pull data from the Apex controller using JavaScript remoting. This data is then wrapped into appropriate HTML elements and rendered on the Visualforce page.

Take a look at the following JavaScript file:.

```
function getAlbums(callback) {
    $j('#albumlist').empty();
    CloudtunesController.queryAlbums(function(records, e)
        { showAlbums(records, callback) }, {escape:true});
}
```

- In `getAlbums()`, calls such as `$j('#albumlist').empty()` are an indication of jQuery at work. In this case, jQuery retrieves the HTML element identified by `albumlist`, and clears out the HTML, readying it for results.
- The method then makes a call to the Apex controller's `queryAlbums()` method. This is where the JavaScript remoting magic happens. Visualforce provides all the required plumbing to allow the call to the controller method directly from the JavaScript.
- Finally, a callback function is passed as an argument to `queryAlbums()` that is automatically invoked once the records are returned from the Apex controller. The `showAlbums()` function takes these records and displays them.

Now let's take a look at `showAlbums()`.

```
function showAlbums(records, callback) {
   currentAlbums.length = 0;
   for(var i = 0; i < records.length; i++) { currentAlbums[records[i].Id] = records[i]; }

   $j.each(records, function() {
      $j('<li></li>')
      .attr('id',this.Id)
      .hide()
      .append('<h2>' + this.Name + '</h2>')
      .click(function(e) {
         e.preventDefault();
         $j.mobile.showPageLoadingMsg();
         $j('#AlbumName').html(currentAlbums[this.id].Name);
         $j('#AlbumPrice').html('$'+currentAlbums[this.id].Price__c);
         if($j('#AlbumPrice').html().indexOf(".") > 0
               && $j('#AlbumPrice').html().split(".")[1].length == 1) {
```

```
            $j('#AlbumPrice').html($j('#AlbumPrice').html()+"0"); //add trailing zero
    }
         $j('#AlbumId').val(currentAlbums[this.id].Id);
         var onTracksLoaded = function() {
         $j.mobile.hidePageLoadingMsg();
         $j.mobile.changePage('#detailpage', {changeHash: true});
    }
         getTracks(currentAlbums[this.id].Id, onTracksLoaded);
    })
     .appendTo('#albumlist')
     .show();
     });

   $j('#albumlist').listview('refresh');
   if(callback != null && typeof callback == 'function') { callback(); }
}
```

- This function gets the records from the callback, loops through them, and creates a new list of `<li>` HTML elements to display within the `albumlist` div.
- Notice this function also dynamically attaches a new event to each list item so that when the user clicks the list item, they can browse down to a list of tracks associated with the album. The list of those tracks is fetched using `getTracks()` .

Now let's take a look at `getTracks()`. Functionally, this code is very similar to the `getAlbums()` and `showAlbums()` code. The only significant difference to the code that handled albums is that a different Apex controller method is used, and of course, a different callback function is provided for updating the page with the results.

```
function getTracks(albumid, callback) {
  $j('#tracklist').empty();
  CloudtunesController.queryTracks(albumid, function(records, e) {
    showTracks(records,callback) }, {escape:true} );
  return true;
}
```

Now anytime the album name is clicked, a new set of track data will be retrieved and the `itemlist` will be rewritten. Clicking on the track name will rewrite the HTML of the elements displaying the track information and use jQuery Mobile to move to that page. A real application can, of course, cache this information as well.

Chapter 7

Bootstrap Events

In this chapter ...

Want to put your personal stamp on your mobile app's startup sequence? Use bootstrap event notifications. Bootstrap events let you customize the messages that your users see while the application is initializing.

Bootstrap Initialization Sequence

When a Mobile SDK hybrid application starts, the first page displayed is bootstrap.html. This page resides in the project's `www` folder. It initializes necessary libraries and plugins, and attempts to connect to the network. The bootstrap process follows this sequence:

1. Load `bootstrap.html`. By default, when the debug flag is set, this file displays console messages generated by the loading process.
2. Load Cordova (PhoneGap) libraries.
3. Load the OAuth plugin.
4. Attempt a network connection. If the connection succeeds, get the name of the server instance. If the connection fails, branch based on the app's configuration.
5. If connected, load the OAuth login screen.

Bootstrap.html and Bootstrap Status Events

Besides initialization tasks, bootstrap.html receives and handles events that report on how the startup operation is faring. The Mobile SDK covers the startup sequence with these bootstrap event types:

Event Type	Description
`AUTHENTICATING`	Sent when the `authenticate()` method is called with the user's login.
`STARTING`	Sent after the user has been authenticated and the application is ready to begin loading. Indicates that the authorization process succeeded.
`OFFLINE`	Sent when no network connection is available.

Bootstrap events are defined in `external/shared/PhoneGap/util/SFHybridApp.js` on the `salesforce/util/event` object. The event object supports three properties:

Property	Description
`code`	The integer identifier for this event type
`description`	A human-friendly message to be consumed by the event handler
`isError`	A Boolean value indicating whether this event resulted in an error condition

Note: Do not modify the `external/shared/PhoneGap/util/SFHybridApp.js` file!

Using Bootstrap Event Notifications

To provide your custom responses to these events, add event handlers to the document in `bootstrap.html` and listen for `bootstrapStatusEvent`. Event handlers are added by calling `addEventListener()` on the HTML DOM document object. Be sure to check the `isError` property for each event: If it's true, take whatever steps are appropriate to handle the fault. In the default implementation, any error condition displays the error message in an alert box and then reloads `bootstrap.html`.

The `handleStatus()` function in `bootstrap.html` provides a generic check for errors, displays the event description, and reloads the page in the event of error. It doesn't differentiate between the various event types. This is where you can provide your custom event handling code. For example, you could display the event descriptions in a custom pop-up. Here's a simple example of a bootstrap event handler that differentiates event types.

```
document.addEventListener("bootstrapStatusEvent", handleStatus);
function handleStatus(statusEvent) {
/**
 * If an error occurs, display a popup, and reload the page.
 */
if (statusEvent.isError) {
  alert(statusEvent.description);
  location.reload();
} else {
  // Display the status in a div element
  var statusDiv = document.getElementById("statusText");
  statusDiv.innerHTML = statusEvent.description;
  switch(statusEvent.EventType)
  {
    case AUTHENTICATING:
        alert("Authenticating...");
        break;
    case STARTING:
        alert("Starting...");
        break;
    case OFFLINE:
        alert("Offline...");
        break;
    default:
        break;
  }
 }
}
```

Chapter 8

Securely Storing Data Offline

In this chapter ...

Mobile devices can lose connection at any time, and environments such as hospitals and airplanes often prohibit connectivity. To handle these situations, it's important that your mobile apps continue to function when they go offline.

The Mobile SDK uses SmartStore, a secure offline storage solution on your device. SmartStore allows you to continue working even when the device is not connected to the Internet. SmartStore stores data as JSON documents in a data structure called a *soup*. A soup is a simple one-table database of "entries" which can be indexed in different ways and queried by a variety of methods.

Note: Pure HTML5 apps store offline information in a browser cache. Browser caching isn't part of the Mobile SDK, and we don't document it here. SmartStore uses storage functionality on the device. This strategy requires a native or hybrid development path.

Sample Objects

The code snippets in this chapter use two objects, Account and Opportunity, which come predefined with every Salesforce organization. Account and Opportunity have a master-detail relationship; an Account can have more than one Opportunity.

Accessing SmartStore in Hybrid Apps

Hybrid containers access native device functionality, such as the camera, address book, and file storage, through JavaScript. SmartStore is also accessed from JavaScript. In order to enable offline access in a hybrid mobile application, you need to include a couple of JavaScript and CSS files in your Visualforce or HTML page.

- `cordova-x.x.x.js` — The Cordova library (formerly PhoneGap).
- `SFHybridApp.js` – Contains methods that perform utility tasks, such as determining whether you're offline.
- `SFSmartStorePlugin.js` – The core implementation of the SDK's offline functionality.

You store your offline data in SmartStore in one or more *soups*. A soup, conceptually speaking, is a logical collection of data records—represented as JSON objects—that you want to store and query offline. In the Force.com world, a soup will typically map to a standard or custom object that you wish to store offline, but that is not a hard and fast rule. You can store as many soups as you want in an application, but remember that soups are meant to be self-contained data sets; there is no direct correlation between them. In addition to storing the data itself, you can also specify indices that map to fields within the data, for greater ease and customization of data queries.

Note:

SmartStore data is inherently volatile. Its lifespan is tied to the authenticated user as well as to OAuth token states. When the user logs out of the app, SmartStore deletes all soup data associated with that user. Similarly, when the OAuth refresh token is revoked or expires, the user's app state is reset, and all data in SmartStore is purged. Carefully consider the volatility of SmartStore data when designing your app. This warning is especially important if your org sets a short lifetime for the refresh token.

Adding SmartStore to Android Apps

In previous Android versions of the Mobile SDK, SmartStore was automatically included in all apps. In the Salesforce Mobile SDK for Android, v. 1.4, SmartStore becomes an optional component. This switch to optional does not apply to the Mobile SDK for iOS.

To use SmartStore in version 1.4 or later, you need to configure your project to include it. When you create a new Android project from the command line, you can do this by adding the optional `–Duser.smartstore=true` parameter. See Creating a New Android Project on page 27 for examples.

To add SmartStore to an existing Android project (hybrid or native) created with Mobile SDK 1.4 or later:

1. Add the SmartStore library project to your project. In Eclipse, choose Properties from the Project menu. Select Android from the left panel, then click Add on the right-hand side. Choose the hybrid/SmartStore project.
2. In your *projectname*App.java file, import the ForceAppWithSmartStore class instead of ForceApp. Replace this statement:

   ```
   import com.salesforce.androidsdk.app.ForceApp
   ```

   with this one:

   ```
   import com.salesforce.androidsdk.app.ForceAppWithSmartStore
   ```

3. In your *projectname*App.java file, change your App class to extend the ForceAppWithSmartStore class rather than ForceApp.

Offline Hybrid Development

Developing a hybrid application inside the container requires a build/deploy step for every change. For that reason, we recommend you develop your hybrid application directly in a browser, and only run your code in the container in the final stages of testing. JavaScript development in a browser is easier because there is no build/compile step. Whenever you make changes to the code, you can refresh the browser to see your changes.

We recommend using the Google Chrome browser because it comes bundled with developer tools that let you access the internals of the your web applications. For more information, see Chrome Developer Tools: Overview.

Using the Mock SmartStore

To facilitate developing and testing code that makes use of the SmartStore while running outside the container, you can use an emulated SmartStore. The MockSmartStore is a JavaScript implementation of the SmartStore that stores the data in local storage (or optionally just in memory).

Note: The MockSmartStore doesn't encrypt data and is not meant to be used in production applications.

Inside the `PhoneGap` directory, there's a local directory containing the following files:

- `MockCordova.js`—A minimal implementation of Cordova functions meant only for testing plugins outside the container.
- `MockSmartStore.js`—A JavaScript implementation of the SmartStore meant only for development and testing outside the container.
- `MockSmartStorePlugin.js`—A JavaScript helper class that intercepts SmartStore Cordova plugin calls and handles them using a MockSmartStore.
- `CordovaInterceptor.js`—A JavaScript helper class that intercepts Cordova plugin calls.

When writing an application using SmartStore, make the following changes to test your app outside the container:

- Include `MockCordova.js` instead of `cordova-x.x.x.js`.
- Include `MockSmartStore.js` after `SFSmartStorePlugin.js`.

To see a MockSmartStore example, check out `Cordova/local/test.html`.

Same-origin Policies

Same-origin policy permits scripts running on pages originating from the same site to access each other's methods and properties with no specific restrictions; it also blocks access to most methods and properties across pages on different sites. Same-origin policy restrictions are not an issue when your code runs inside the container, because the container disables same-origin policy in the webview. However, if you call a remote API, you need to worry about same-origin policy restrictions.

Fortunately, browsers offer ways to turn off same-origin policy, and you can research how to do that with your particular browser. If you want to make XHR calls against Force.com from JavaScript files loaded from the local file system, you should start your browser with same-origin policy disabled. The following article describes how to disable same-origin policy on several popular browsers: Getting Around Same-Origin Policy in Web Browsers.

Authentication

For authentication with MockSmartStore, you will need to capture access tokens and refresh tokens from a real session and hand code them in your JavaScript app. You'll also need these tokens to initialize the `ForceTk` JavaScript toolkit.

Registering a Soup

In order to access a soup, you first need to register it. Provide a name, index specifications, and names of callback functions for success and error conditions:

```
navigator.smartstore.registerSoup(soupName, indexSpecs, successCallback, errorCallback)
```

If the soup does not already exist, this function creates it. If the soup already exists, registering gives you access to the existing soup. To find out if a soup already exists, use:

```
navigator.smartstore.soupExists(soupName, successCallback, errorCallback);
```

A soup is indexed on one or more fields found in its entries. Insert, update, and delete operations on soup entries are tracked in the soup indices. Always specify at least one index field when registering a soup. For example, if you are using the soup as a simple key/value store, use a single index specification with a string type.

indexSpecs

The `indexSpecs` array is used to create the soup with predefined indexing. Entries in the `indexSpecs` array specify how the soup should be indexed. Each entry consists of a `path:type` pair. `path` is the name of an index field; `type` is either "string" or "integer". Index paths are case-sensitive and can include compound paths, such as Owner.Name.

Note: Performance can suffer if the index path is too deep. If index entries are missing any fields described in a particular `indexSpec`, they will not be tracked in that index.

```
 "indexSpecs":[
    {
        "path":"Name",
        "type":"string"
    }
    {
        "path":"Id",
        "type":"string"
    }
    {
        "path":"ParentId",
        "type":"string"
    }
    {
        "path":"lastModifiedDate",
        "type":"integer"
    }
]
```

Note: Currently, the Mobile SDK supports two index types: "string" and "integer." These types apply only to the index itself, and not to the way data is stored or retrieved. It's OK to have a null field in an index column.

`successCallback`

The success callback function for `registerSoup` takes one argument (the soup name).

```
function(soupName) { alert("Soup " + soupName + " was successfully created"); };
```

A successful creation of the soup returns a `successCallback` that indicates the soup is ready. Wait to complete the transaction and receive the callback before you begin any activity. If you register a soup under the passed name, the success callback function returns the soup.

`errorCallback`

The error callback function for `registerSoup` takes one argument (the error description string).

```
function(err) { alert ("registerSoup failed with error:" + err); }
```

During soup creation, errors can happen for a number of reasons, including:

- An invalid or bad soup name
- No index (at least one index must be specified)
- Other unexpected errors, such as a database error

Retrieving Data From a Soup

SmartStore provides a set of helper methods that build query strings for you. To query a specific set of records, call the `build*` method that suits your query specification. You can optionally define the index field, sort order, and other metadata to be used for filtering, as described in the following table:

Parameter	Description
`indexPath`	This is what you're searching for; for example a name, account number, or date.
`beginKey`	Optional. Used to define the start of a range query.
`endKey`	Optional. Used to define the end of a range query.
`order`	Optional. Either "ascending" or "descending."
`pageSize`	Optional. If not present, the native plugin can return whatever page size it sees fit in the resulting `Cursor.pageSize`.

Note:

All queries are single-predicate searches. Queries don't support joins.

Query Everything

`buildAllQuerySpec(indexPath, order, [pageSize])` returns all entries in the soup, with no particular order. Use this query to traverse everything in the soup.

`order` and `pageSize` are optional, and default to ascending and 10, respectively. You can specify:

- `buildAllQuerySpec(indexPath)`
- `buildAllQuerySpec(indexPath, order)`
- `buildAllQuerySpec(indexPath, order, [pageSize])`

However, you can't specify `buildAllQuerySpec(indexPath,[pageSize])`.

See Working With Cursors for information on page sizes.

Note: As a base rule, set `pageSize` to the number of entries you want displayed on the screen. For a smooth scrolling display, you might want to increase the value to two or three times the number of entries actually shown.

Query by Exact

`buildExactQuerySpec(indexPath, matchKey, [pageSize])` finds entries that exactly match the given `matchKey` for the `indexPath` value. Use this to find child entities of a given ID. For example, you can find Opportunities by Status. However, you can't specify order in the results.

Sample code for retrieving children by ID:

```
var querySpec = navigator.smartstore.buildExactQuerySpec("sfdcId", "some-sfdc-id");
navigator.smartstore.querySoup("Catalogs", querySpec, function(cursor) {
       // we expect the catalog to be in: cursor.currentPageOrderedEntries[0]
});
```

Sample code for retrieving children by parent ID:

```
var querySpec = navigator.smartstore.buildExactQuerySpec("parentSfdcId", "some-sfdc-id);
navigator.smartstore.querySoup("Catalogs", querySpec, function(cursor) {});
```

Query by Range

`buildRangeQuerySpec(indexPath, beginKey, endKey, [order, pageSize])` finds entries whose `indexPath` values fall into the range defined by `beginKey` and `endKey`. Use this function to search by numeric ranges, such as a range of dates stored as integers.

`order` and `pageSize` are optional, and default to ascending and 10, respectively. You can specify:

- `buildRangeQuerySpec(indexPath, beginKey, endKey)`
- `buildRangeQuerySpec(indexPath, beginKey, endKey, order)`
- `buildRangeQuerySpec(indexPath, beginKey, endKey, order, pageSize)`

However, you can't specify `buildRangeQuerySpec(indexPath, beginKey, endKey, pageSize)`.

By passing null values to `beginKey` and `endKey`, you can perform open-ended searches:

- Passing `null` to `endKey` finds all records where the field at `indexPath` is >= `beginKey`.
- Passing `null` to `beginKey` finds all records where the field at `indexPath` is <= `endKey`.
- Passing `null` to both `beginKey` and `endKey` is the same as querying everything.

Query by Like

`buildLikeQuerySpec(indexPath, likeKey, [order, pageSize])` finds entries whose `indexPath` values are like the given `likeKey`. You can use "foo%" to search for terms that begin with your keyword, "%foo" to search for terms that end with your keyword, and "%foo%" to search for your keyword anywhere in the `indexPath` value. Use this function for general searching and partial name matches. `order` and `pageSize` are optional, and default to ascending and 10, respectively.

Note: Query by Like is the slowest of the query methods.

Executing the Query

Queries run asynchronously and return a cursor to your JavaScript callback. Your success callback should be of the form `function(cursor)`. Use the `querySpec` parameter to pass your query specification to the `querySoup` method.

```
navigator.smartstore.querySoup(soupName,querySpec,successCallback,errorCallback);
```

Retrieving Individual Soup Entries by Primary Key

All soup entries are automatically given a unique internal ID (the primary key in the internal table that holds all entries in the soup). That ID field is made available as the `_soupEntryId` field in the soup entry. Soup entries can be looked up by `_soupEntryId` by using the `retrieveSoupEntries` method. Note that the return order is not guaranteed, and if entries have been deleted they will be missing from the resulting array. This method provides the fastest way to retrieve a soup entry, but it's usable only when you know the `_soupEntryId`:

```
navigator.smartStore.retrieveSoupEntries(soupName, indexSpecs, successCallback, errorCallback)
```

Working With Cursors

Queries can potentially have long result sets that are too large to load. Instead, only a small subset of the query results (a single page) is copied from the native realm to the JavaScript realm at any given time. When you perform a query, a cursor object is returned from the native realm that provides a way to page through a list of query results. The JavaScript code can then move forward and back through the pages, causing pages to be copied to the JavaScript realm.

Note: For advanced users: Cursors are not snapshots of data; they are dynamic. If you make changes to the soup and then start paging through the cursor, you will see those changes. The only data the cursor holds is the original query and your current position in the result set. When you move your cursor, the query runs again. Thus, newly created soup entries can be returned (assuming they satisfy the original query).

Use the following cursor functions to navigate the results of a query:

- `navigator.smartstore.moveCursorToPageIndex(cursor, newPageIndex, successCallback, errorCallback)`—Move the cursor to the page index given, where 0 is the first page, and the last page is defined by `totalPages - 1`.
- `navigator.smartstore.moveCursorToNextPage(cursor, successCallback, errorCallback)`—Move to the next entry page if such a page exists.
- `navigator.smartstore.moveCursorToPreviousPage(cursor, successCallback, errorCallback)`—Move to the previous entry page if such a page exists.
- `navigator.smartstore.closeCursor(cursor, successCallback, errorCallback)`—Close the cursor when you're finished with it.

Note: `successCallback` for those functions should expect one argument (the updated cursor).

Manipulating Data

In order to track soup entries for insert, update, and delete, the SmartStore adds a few fields to each entry:

- `_soupEntryId`—This field is the primary key for the soup entry in the table for a given soup.
- `_soupLastModifiedDate`—The number of milliseconds since 1/1/1970.
  - ◊ To convert to a JavaScript date, use `new Date(entry._soupLastModifiedDate)`
  - ◊ To convert a date to the corresponding number of milliseconds since 1/1/1970, use `date.getTime()`

When inserting or updating soup entries, SmartStore automatically sets these fields. When removing or retrieving specific entries, you can reference them by `_soupEntryId`.

Inserting or Updating Soup Entries

If the provided soup entries already have the `_soupEntryId` slots set, then entries identified by that slot are updated in the soup. If an entry does not have a `_soupEntryId` slot, or the value of the slot doesn't match any existing entry in the soup, then the entry is added (inserted) to the soup, and the `_soupEntryId` slot is overwritten.

Note: You must not manipulate the `_soupEntryId` or `_soupLastModifiedDate` value yourself.

Use the `upsertSoupEntries` method to insert or update entries:

```
navigator.smartStore.upsertSoupEntries(soupName, entries[], successCallback, errorCallback)
```

where `soupName` is the name of the target soup, and `entries` is an array of one or more entries that match the soup's data structure. The `successCallback` and `errorCallback` parameters function much like the ones for `registerSoup`. However, the success callback for `upsertSoupEntries` indicates that either a new record has been inserted, or an existing record has been updated.

Upserting with an External ID

If your soup entries mirror data from an external system, you might need to refer to those entities by their ID (primary key) in the external system. For that purpose, we support upsert with an external ID. When you perform an upsert, you can designate any index field as the external ID field. SmartStore will look for existing soup entries with the same value in the designated field with the following results:

- If no field with the same value is found, a new soup entry will be created.
- If the external ID field is found, it will be updated.
- If more than one field matches the external ID, an error will be returned.

When creating a new entry locally, use a regular upsert. Set the external ID field to a value that you can later query when uploading the new entries to the server.

When updating entries with data coming from the server, use the upsert with external ID. Doing so guarantees that you don't end up with duplicate soup entries for the same remote entity.

In the following sample code, we chose the value `new` for the `id` field because the record doesn't yet exist on the server. Once we are online, we can query for records that exist only locally (by looking for records where `id == "new"`) and upload them to the server. Once the server returns the actual ID for the records, we can update their `id` fields locally. If you create products that belong to catalogs that have not yet been created on the server, you will be able to capture the relationship with the catalog through the `parentSoupEntryId` field. Once the catalogs are created on the server, update the local records' `parentExternalId` fields.

The following code contains sample scenarios. First, it calls `upsertSoupEntries` to create a new soup entry. In the success callback, the code retrieves the new record with its newly assigned soup entry ID. It then changes the description and calls `ForceTK` methods to create the new account on the server and then update it. The final call demonstrates the upsert with external ID. To make the code more readable, no error callbacks are specified. Also, because all SmartStore calls are asynchronous, real applications should do each step in the callback of the previous step.

```
// Specify data for the account to be created
var acc = {id: "new", Name: "Cloud Inc", Description: "Getting started"};

// Create account in SmartStore
// This upsert does a "create" because the acc has no _soupEntryId field
navigator.smartstore.upsertSoupEntries("accounts", [ acc ], function(accounts) {
    acc = accounts[0];
    // acc should now have a _soupEntryId field (and a _lastModifiedDate as well)
});

// Update account's description in memory
acc["Description"] = "Just shipped our first app ";

// Update account in SmartStore
// This does an "update" because acc has a _soupEntryId field
navigator.smartstore.upsertSoupEntries("accounts", [ acc ], function(accounts) {
   acc = accounts[0];
});

// Create account on server (sync client -> server for entities created locally)
forcetkClient.create("account", {"Name": acc["Name"], "Description": acc["Description"]},
function(result) {
   acc["id"] = result["id"];
   // Update account in SmartStore
   navigator.smartstore.upsertSoupEntries("accounts", [ acc ]);
});

// Update account's description in memory
acc["Description"] = "Now shipping for iOS and Android";

// Update account's description on server
// Sync client -> server for entities existing on server
forcetkClient.update("account", acc["id"], {"Description": acc["Description"]});

///// Later, there is an account (with id: someSfdcId) that you want to get locally

///// There might be an older version of that account in the SmartStore already

// Update account on client
// sync server -> client for entities that might or might not exist on client
forcetkClient.retrieve("account", someSfdcId, "id,Name,Description", function(result) {
   // Create or update account in SmartStore (looking for an account with the same sfdcId)
   navigator.smartstore.upsertSoupEntriesWithExternalId("accounts", [ result ], "id");
});
```

Removing Soup Entries

Entries are removed from the soup asynchronously and your callback is called with success or failure. The `soupEntryIds` is a list of the `_soupEntryId` values from the entries you wish to delete.

```
navigator.smartStore.removeFromSoup(soupName, soupEntryIds, successCallback, errorCallback)
```

Removing a Soup

To remove a soup, call `removeSoup()`. Note that once a user signs out, the soups get deleted automatically.

```
navigator.smartstore.removeSoup(soupName,successCallback,errorCallback);
```

SmartStore Extensions

Some apps might profit by extending the SmartStore API in various ways.

- Secure localStorage—W3C's `localStorage` is a simple key-value storage that can be readily implemented on top of SmartStore. For instance, a single `localStorage` soup can be created by the SmartStore plugin on each platform, and the `matchKey` can be the key passed to the `localStorage` methods. This is a convenience layer for developers who are already familiar with `localStorage` and comfortable with its limitations. The main difference in our implementation is the need to rely on Cordova-style JavaScript callbacks, so all `localStorage` methods are asynchronous.
- Files and Large Binary Objects—Some apps require the ability to store large binary objects, such as video, PDF, and PPT files. For these apps, there is currently no consistent secure storage mechanism in Cordova.

Chapter 9

Migrating from the Previous Release

In this chapter ...

If you developed code with Salesforce Mobile SDK 1.4, follow these instructions to update your app to version 1.5.

Refresh Token Revocation (Android Native Apps)

Salesforce Mobile SDK 1.5 lets an Android native app decide how to react when an administrator revokes a user's refresh token. In previous releases, the app had no control over these events. In version 1.5, you can provide your own event handler if your app needs to override the default response. See Handling Refresh Token Revocation in Android Native Apps on page 17.

Note: FOR ALL LEGACY ANDROID NATIVE APPS: Minor code changes are required if your app contains activities that do not extend `com.salesforce.androidsdk.ui.NativeMainActivity`. This requirement applies even if you don't override the default token revocation behavior. See Handling Refresh Token Revocation in Android Native Apps on page 17.

Update Cordova Libraries

Salesforce Mobile SDK 1.5 updates Cordova support from version 2.2 to version 2.3. If your project references the Cordova library anywhere in your own app, be sure you update your code and configuration accordingly.

See Also:

Versioning and Javascript Library Compatibility
Example: Serving the Appropriate Javascript Libraries

Chapter 10

Reference

In this chapter ...

Reference documentation is hosted on GitHub

- For iOS: http://forcedotcom.github.com/SalesforceMobileSDK-iOS/Documentation/SalesforceSDK/index.html
- For Android: http://forcedotcom.github.com/SalesforceMobileSDK-Android/index.html

REST API Resources

The Salesforce Mobile SDK simplifies using the REST API by creating wrappers. All you need to do is call a method and provide the correct parameters; the rest is done for you. This table lists the resources available and what they do. For more information, see the REST API Developer's Guide.

Resource Name	URI	Description
Versions	`/`	Lists summary information about each Salesforce version currently available, including the version, label, and a link to each version's root.
Resources by Version	`/vXX.X/`	Lists available resources for the specified API version, including resource name and URI.
Describe Global	`/vXX.X/sobjects/`	Lists the available objects and their metadata for your organization's data.
SObject Basic Information	`/vXX.X/sobjects/`***`SObject`***`/`	Describes the individual metadata for the specified object. Can also be used to create a new record for a given object.
SObject Describe	`/vXX.X/sobjects/`***`SObject`***`/describe/`	Completely describes the individual metadata at all levels for the specified object.
SObject Rows	`/vXX.X/sobjects/`***`SObject`***`/`***`id`***`/`	Accesses records based on the specified object ID. Retrieves, updates, or deletes records. This resource can also be used to retrieve field values.
SObject Rows by External ID	`/vXX.X/sobjects/`***`SObjectName`***`/`***`fieldName`***`/`***`fieldValue`***	Creates new records or updates existing records (upserts records) based on the value of a specified external ID field.
SObject Blob Retrieve	`/vXX.X/sobjects/`***`SObject`***`/`***`id`***`/`***`blobField`***	Retrieves the specified blob field from an individual record.
SObject User Password	`/vXX.X/sobjects/User/`***`user id`***`/password` `/vXX.X/sobjects/SelfServiceUser/`***`self service user id`***`/password`	Set, reset, or get information about a user password.
Query	`/vXX.X/query/?q=`***`soql`***	Executes the specified SOQL query.
Search	`/vXX.X/search/?s=`***`sosl`***	Executes the specified SOSL search. The search string must be URL-encoded.

iOS Architecture

At a high level, the current facilities that the native SDK provides to consumers are:

- OAuth authentication capabilities
- REST API communication capabilities
- SmartStore secure storage and retrieval of app data

Note: This is not currently exposed to native template apps, but is included in the binary distribution.

The Salesforce native SDK is essentially one library, with dependencies on (and providing exposure to) the following additional libraries:

- `libRestKit.a` — Third-party underlying libraries for facilitating REST API calls.
    - ◊ RestKit in turn depends on `libxml2.dylib`, which is part of the standard iOS development environment
- `libSalesforceOAuth.a` — Underlying libraries for managing OAuth authentication.
- `libsqlite3.dylib` — Library providing access to SQLite capabilities. This is also a part of the standard iOS development environment.
- `fmdb` — Objective-C wrapper around SQLite.

Note: This is not currently exposed to native template apps, but exposed in the binary distribution.

Native iOS Objects

The following objects are important for leveraging Mobile SDK functionality in your app:

- `SFRestAPI`
- `SFRestAPI` (Blocks)
- `SFRestRequest`

SFRestAPI

`SFRestAPI` is the entry point for making REST requests, and is generally accessed as a singleton, via `SFRestAPI sharedInstance`.

You can easily create many standard canned queries from this object, such as:

```
SFRestRequest* request = [[SFRestAPI sharedInstance] requestForUpdateWithObjectType:@"Contact"

    objectId:contactId
    fields:updatedFields];
```

You can then initiate the request with the following:

```
[[SFRestAPI sharedInstance] send:request delegate:self];
```

`SFRestAPI` (Blocks)

This is a category extension of the `SFRestAPI` class that allows you to specify blocks as your callback mechanism. For example:

```
NSMutableDictionary *fields = [NSMutableDictionary dictionaryWithObjectsAndKeys:
    @"John", @"FirstName",
    @"Doe", @"LastName",
    nil];
[[SFRestAPI sharedInstance] performCreateWithObjectType:@"Contact"
    fields:fields
    failBlock:^(NSError *e) {
    NSLog(@"Error: %@", e);
    }
    completeBlock:^(NSDictionary *d) {
        NSLog(@"ID value for object: %@", [d objectForKey:@"id"]);
    }];
```

`SFRestRequest`

In addition to the canned REST requests provided by SFRestAPI, you can also create your own:

```
NSString *path = @"/v23.0";
SFRestRequest* request = [SFRestRequest requestWithMethod:SFRestMethodGET path:path
queryParams:nil];
```

Other Objects

Though you won't likely leverage these objects directly, their purpose in the SDK is worth noting.

- `RKRequestDelegateWrapper`—The intermediary between `SFRestAPI` and the `RestKit` libraries. `RKRequestDelegateWrapper` wraps the functionality of `RestKit` communications, providing convenience methods for determining the type of HTTP post, handling data transformations, and interpreting responses.
- `SFSessionRefresher`—Tightly-coupled with `SFRestAPI`, providing an abstraction around functionality for automatically refreshing a session if any REST requests fail due to session expiration.

Android Architecture

The SalesforceSDK is provided as a library project. You need to reference the SalesforceSDK project from your application project. See the Android developer documentation.

Java Code

Java sources are under `/src`.

Java Code

Package Name	Description
`com.salesforce.androidsdk.app`	SDK application classes (ForceApp)
`com.salesforce.androidsdk.auth`	OAuth support classes
`com.salesforce.androidsdk.phonegap`	Native implementation of Salesforce Mobile SDK PhoneGap plugin

Package Name	Description
`com.salesforce.androidsdk.rest`	Classes for REST requests/responses
`com.salesforce.androidsdk.security`	Security-related helper classes (e.g. passcode manager)
`com.salesforce.androidsdk.store`	SmartStore and supporting classes
`com.salesforce.androidsdk.ui`	Activities (e.g. login)
`com.salesforce.androidsdk.util`	Miscellaneous utility classes

`com.salesforce.androidsdk.app`

Class	Description
`ForceApp`	Abstract subclass of application; you must supply a concrete subclass in your project.
`UpgradeManager`	Helper class for upgrades
`UUIDManager`	Helper class for UUID generation
`ForceAppWithSmartStore`	Super class for all force applications that use the SmartStore (lives in SmartStore library project)
`UpgradeManagerWithSmartStore`	Upgrade manager for applications that use the SmartStore (lives in SmartStore library project)

`com.salesforce.androidsdk.auth`

Class	Description
`AccountWatcher`	Watcher responsible for cleanup when account is removed from settings application
`AuthenticatorService`	Service taking care of authentication
`HttpAccess`	Generic HTTP access layer
`LoginServerManager`	Manages login hosts
`OAuth2`	Helper class for common OAuth2 requests

`com.salesforce.androidsdk.phonegap`

Class	Description
`ForcePlugin`	Abstract super class for all Salesforce plugins
`JavaScriptPluginVersion`	Helper class to encapsulate the version reported by the JavaScript code
`SalesforceOAuthPlugin`	PhoneGap plugin for Salesforce OAuth
`SmartStorePlugin`	PhoneGap plugin for SmartStore (lives in the SmartStore library project)

Class	Description
`SDKInfoPlugin`	PhoneGap plugin to get information about the SDK container
`TestRunnerPlugin`	PhoneGap plugin to run javascript tests in container

`com.salesforce.androidsdk.rest`

Class	Description
`ClientManager`	Factory of RestClient, kicks off login flow if needed
`RestClient`	Authenticated client to talk to a Force.com server
`RestRequest`	Force.com REST request wrapper
`RestResponse`	REST response wrapper

`com.salesforce.androidsdk.security`

Class	Description
`Encryptor`	Helper class for encryption/decryption/hash computations
`PasscodeManager`	Inactivity timeout manager, kicks off passcode screen if needed

`com.salesforce.androidsdk.store`

This package is part of the SmartStore library project.

Class	Description
`DBOpenHelper`	Helper class to manage regular database creation and version management
`DBOperations`	`DBOpenHelper/EncryptedDBOpenHelper` wrapper
`SmartStore`	Searchable/secure store for JSON documents

`com.salesforce.androidsdk.ui`

Class	Description
`CustomServerUrlEditor`	Custom dialog allowing user to pick a different login host
`LoginActivity`	Login screen
`NativeMainActivity`	Main activity of native application should extend or duplicate the functionality of this class
`OAuthWebviewHelper`	Helper class to manage a WebView instance that is going through the OAuth login process
`PasscodeActivity`	Passcode (PIN) screen
`SalesforceDroidGapActivity`	Main activity for hybrid applications

Class	Description
`SalesforceGapViewClient`	WebView client used in hybrid applications
`SalesforceR`	Class that allows references to resources defined outside the SDK
`ServerPickerActivity`	Choose login host screen

`com.salesforce.androidsdk.util`

Class	Description
`BaseActivityInstrumentationTestCase`	Super class for activty test classes
`EventsListenerQueue`	Class to track activity events using a queue, allowing for tests to wait for certain events to turn up
`EventsObservable`	Used to register and receive events generated by the SDK (used primarily in tests)
`EventsObserver`	Observer of SDK events
`ForceAppInstrumentationTestCase`	Super class for tests of an application using the Salesforce Mobile SDK
`HybridInstrumentationTestCase`	Super class for tests of hybrid application
`JSTestCase`	Super class to run tests written in JavaScript
`JUnitReportTestRunner`	Test runner that runs tests using a time run cap
`LogUtil`	Helper methods for logging
`NativeInstrumentationTestCase`	Super class for tests of native application
`TimeLimitedTestRunner`	Test runner that limits the lifetime of the test run

Libraries

Libraries are under `/libs`.

Library Name	Description
`cordova-2.3.0.jar`	Open source mobile development framework; used in hybrid applications (*)
`sqlcipher.jar`	Open source extension to SQLite that provides transparent 256-bit AES encryptiong of database files (**)
`x86/*.so`	Native libraries required by sqlcipher on Intel-based devices
`armeabi/*.so`	Native libaries required by sqlcipher on ARM-based devices (**)
`commons-code.jar, guava-r09.jar`	Java libraries required by sqlcipher

(*) denotes files required for hybrid application.

(**) denotes files required for SmartStore.

Resources

Resources are under `/res`.

`drawable-hdpi, drawable-ldpi`

File	Use
`edit_icon.png`	Server picker screen
`glare.png`	Login screen
`icon.png`	Application icon

`drawable`

File	Use
`header_bg.png`	Login screen
`progress_spinner.xml`	Login screen
`toolbar_background.xml`	Login screen

`drawable-xlarge, drawable-xlarge-port`

File	Use
`header_bg.png`	Login screen (tablet)
`header_drop_shadow.xml`	Login screen (tablet)
`header_left_border.xml`	Login screen (tablet)
`header_refresh.png`	Login screen (tablet)
`header_refresh_press.png`	Login screen (tablet)
`header_refresh_states.xml`	Login screen (tablet)
`header_right_border.xml`	Login screen (tablet)
`login_content_header.xml`	Login screen (tablet)
`nav_shadow.png`	Login screen (tablet)
`oauth_background.png`	Login screen (tablet)
`oauth_container_dropshadow.9.png`	Login screen (tablet)
`oauth_background.png`	Login screen (tablet)
`progress_spinner.xml`	Login screen (tablet)
`refresh_loader.png`	Login screen (tablet)

File	Use
`toolbar_background.xml`	Login screen (tablet)

`layout`

File	Use
`custom_server_url.xml`	Server picker screen
`login.xml`	Login screen
`passcode.xml`	Pin screen
`server_picker.xml`	Server picker screen

`layout-xlarge`

File	Use
`login_header.xml`	Login screen (tablet)
`login.xml`	Login screen (tablet)
`server_picker_header.xml`	Server picker screen (tablet)
`server_picker.xml`	Server picker screen (tablet)

`menu`

File	Use
`clear_custom_url.xml`	Add connection dialog
`login.xml`	Login menu (phone)

`values`

File	Use
`sdk.xml`	Localized strings for login, server picker, and pin screens
`strings.xml`	Other strings (app name)

`values-xlarge`

File	Use
`styles.xml`	Styles (tablet)

`xml`

File	Use
`authenticator.xml`	Preferences for account used by application
`plugins.xml`	Plugin configuration file for PhoneGap. Required for hybrid.

Index