


# ***Mobile SSO and RPC Library***

## Developer Guide



*February 2016*

## Table of Contents

[Background](#)

[Scope](#)

[Glossary](#)

[Components of the SSO Approach](#)

[Single Sign On App](#)

[Sample Application Flow](#)

[Remote Procedure Call Library \(RPCLib\)](#)

[RPCLib API Reference](#)

[Interfaces \(Protocols\)](#)

[IRPCObject](#)

[Methods](#)

[IRPCHandler](#)

[Methods](#)

[RPCLayer](#)

[Methods](#)

[RPCLib Use Cases](#)

[Authentication and Accessing Web Service Data](#)

[Cordova Projects](#)

[Authentication Only](#)

[Application Lifecycle Events](#)

[Application Launch \(mandatory\)](#)

[Cordova Projects](#)

[Application Leaves Foreground \(mandatory for DC2\)](#)

[Integrating the RPCLib into a Project](#)

[IDE Activities](#)

[Configuration](#)

[Cordova Integration](#)

[RPCLib Distribution](#)

[Binary Distribution](#)

[Sample App Distribution](#)

[Appendix](#)

[Keychain Items](#)

[Error Codes](#)

[Detailed Internal Flow](#)

[Revision History](#)

# **1 Background**

## **1.1 Scope**

This document will describe the mobile Single Sign-On (SSO) capabilities available to all territories that adhere to the PwC IT MobileIron device management standard. [Follow this link for further details regarding this standard.](#)

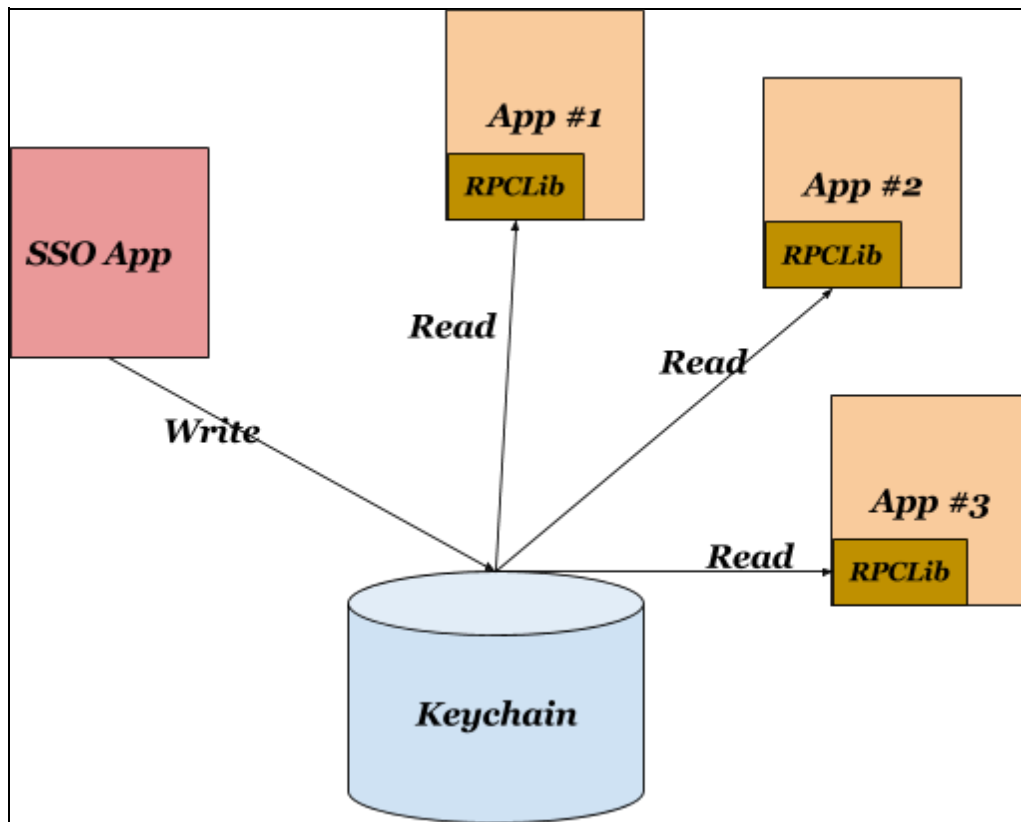
SSO is a core mobile application service that will provide secure authentication and access to native iOS mobile applications on a PwC-managed device. As the mobile strategy and standards for other platforms are defined, SSO capabilities will be offered to other devices as well. The SSO solution provides for the following capabilities, implemented by the two main components:

- Silent user authentication that does not prompt the user for credentials
- Access to data available through secured services

Both are made possible by storing a user-specific X509 certificate, issued by the PwC PKI infrastructure, onto the device in a manner that makes it available for sharing between PwC signed applications.

The following mobile SSO components will be described in detail throughout this document:

- a. Single Sign-On (SSO) Application** is responsible for provisioning a user identity certificate to the iOS device, and is used once by the end user. The SSO App validates the user's identity, then writes the X.509 certificates to the device's keychain.
- b. Remote Procedure Call (RPC) Library** is utilized by native and hybrid iOS applications (global or local territory) to authenticate and authorize the user's identity certificate prior to application usage, and to access protected service data. The RPCLib reads the X.509 certificates from the device's keychain and interfaces with the identity management systems.



## 1.2 Glossary

Term/Abbreviation/Acronym	Definition/Expansion
SM	SiteMinder
SSO	Single Sign On
App	Application
iOS	iPhone Operating System
PKI	Public Key Infrastructure
RPC	Remote Procedure (web service) Calls

## 2 Components of the SSO Approach

### 2.1 Single Sign On App

The Single Sign-On (SSO) application is a native iOS application that is used to obtain the user-specific certificate from PwC's PKI infrastructure and store it on a PwC managed device. A user would only use the SSO application when this certificate is missing or invalid, so it constitutes a one time setup type application. The application prompts the user for credentials (PwC GUID and GUID password) the first time the user launches the application. Once the credentials are validated, the application requests a user identity certificate from the PwC IT mobile certificate authority (CA) and stores it on the device.

The SSO application will store the following information in iOS application keychain:

1. User identity certificate
2. PwC Network certificates (Root, Policy, Issuing1 & Issuing2 or SHA-2 equivalent)
3. Additional attributes: GUID, PPID, Country, Employee ID, email address

The items available in the iOS keychain can be utilized by any application codedesigned with the same Apple Enterprise certificate. The application will need to list access to the PwC [keychain access group](#) in its list of entitlements (see the section on [Integration](#)).

### 2.1.1 Sample Application Flow

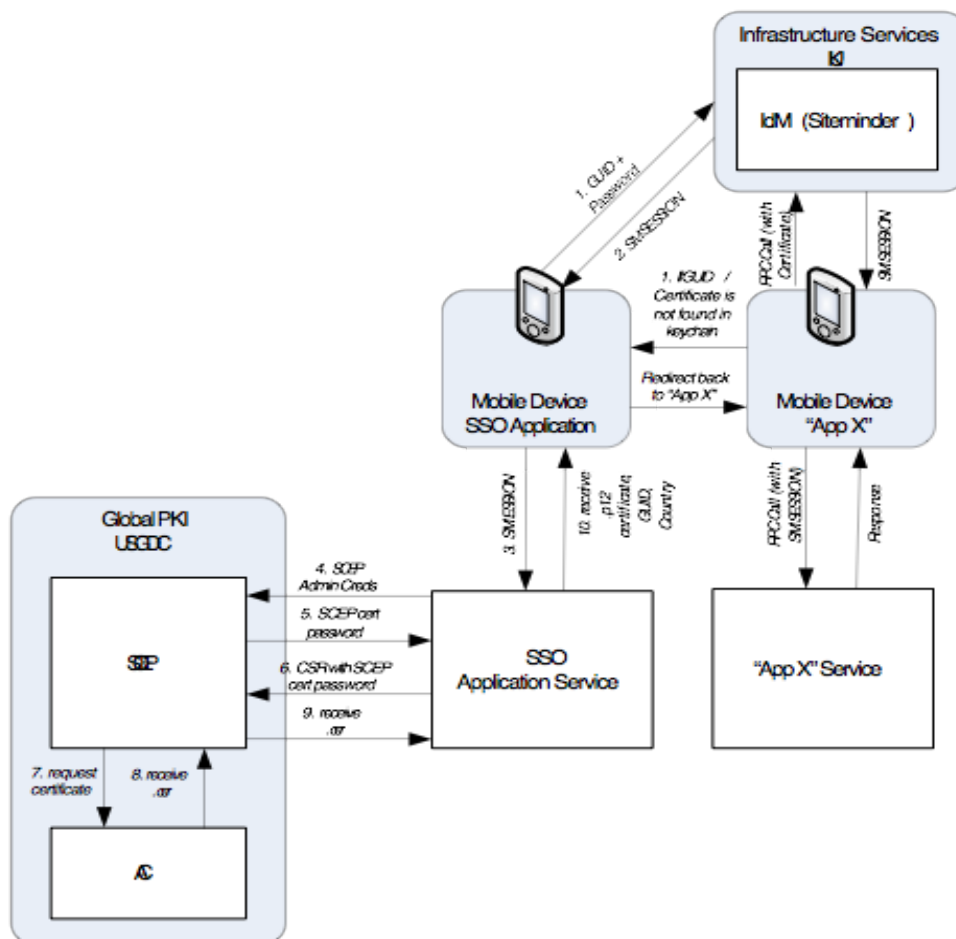


Figure 1 - Sample App Flow

If a certificate is not found in the device keychain, the app should launch the SSO application using the URL scheme approach available in iOS (see the “InvokeSSO” method in the API Reference). Once the certificate is provisioned to the device with the SSO application, the SSO app can invoke a call back to the originating application so that the user never has to manually launch the SSO app in order to be provisioned and gain use of the application. Refer to the description of the “InvokeSSO” method for more details.

For further details on interaction between SSO & Individual iOS apps, refer the detailed flow in section 3.2 (Detailed level flow)

## 2.2 Remote Procedure Call Library (RPCLib)

The RPC Library (RPCLib) provides an Objective-C interface to invoke PwC web services while adhering to PwC IT authentication standards. This library can retrieve data from services that are protected by either SiteMinder or Global IdAM protection. The RPCLib represents a coarse abstraction that encompasses authentication, network connection handling and object serialization/deserialization.

The following are some of the key functions performed by the RPCLib that require no programming effort from client applications:

1. Certificate based authentication with IdM SiteMinder or Global IdAM based on user information stored in iOS device keychain
2. Provides a simplified interface that accepts a url and supporting objects from consumer apps and provides data back to apps
3. Performs retry in case of VPN or other transient connection issues
4. Provides user attribute information to consuming application through iOS keychain handling
5. Provides a way to launch the SSO app if the device is not provisioned

## 3 *RPCLib API Reference*

### 3.1 Interfaces (Protocols)

#### 3.1.1 *IRPCObject*

IRPCObject is a protocol that represents a serializable piece of data to be sent with the web service request. This typically represents POST data that is sent to the web service endpoint. Client applications will create an object that conforms to this protocol when invoking web services using the RPCLib.

##### 3.1.1.1 Methods

@required  
-(NSString \*)serialize:(NSDictionary \*)aDictMsg;

Parameter	Type	Description
aDictMsg	NSDictionary	The data to be serialized
(return)	NSString	The serialized representation. This would be JSON- or XML-formatted data, depending on the requirements of the service endpoint.

A default implementation of the IRPCObject protocol named RPCObject is available for use if the data is easily serialized using iOS's default JSONSerialization process.

#### 3.1.2 *IRPCHandler*

The IRPCHandler represents the delegate (callback) protocol used by the RPCLib to report progress on web service or authentication method calls. Client applications will create an object that conforms to this protocol when invoking web services using the RPCLib, and should use the methods noted here to react to the event lifecycle of a web service call.

### 3.1.2.1 Methods

#### **-(void)onRPCResponseHeaders:(NSMutableDictionary \*)responseHeaders;**

Called when HTTP headers are sent by the server to the RPCLib. This is before the full set of data arrives from the server. This method is most useful when the server sends custom response headers to which the client application should respond. This is not a required protocol method, and is not frequently used.

<i>Parameter</i>	<i>Type</i>	<i>Description</i>
responseHeaders	NSMutableDictionary	The full set of headers, as received by RPCLib. These are identical to those provided by the NSHTTPURLResponse class. Due to the nature of HTTP redirects, clients may receive more than one callback for “onRPCResponseHeaders” for only one request.

#### **-(void)OnSingleSignOnSuccess:(NSString \*)aStrToken;**

Called when the authentication step occurs successfully to the Siteminder endpoint. Represents the final call in the RPCLib lifecycle for a call to “AuthUser”. When calling a service endpoint, this method is most useful in debugging and in most cases is not used.

<i>Parameter</i>	<i>Type</i>	<i>Description</i>
aStrToken	NSString	The value of any new server cookies.

#### **-(void)OnSingleSignOnError:(NSError \*)aError;**

Called by RPCLib when authentication has failed. The app should then call RPCLayer’s “InvokeSSO” method in response. Handling this callback is mandatory.

<i>Parameter</i>	<i>Type</i>	<i>Description</i>
aError	NSError	The cause for the failure.

#### **-(void)OnRPCResponse :(NSData \*)aResponseData :(RPCType )aRPCTypeOfData;**

Called by RPCLib when all web service response data has arrived. Handling this callback is mandatory.

<i>Parameter</i>	<i>Type</i>	<i>Description</i>
aResponseData	NSData	Data from the results of the web service call.
aRPCTypeOfData	RPCType	One of: RPCSOAP_POST, RPCRestfulJSON_POST, RPCSOAP_GET, RPCRestfulJSON_GET, RPCRestfulJSON_DELETE, RPCRestfulJSON_PUT, RPCRestfulJSON_HEAD

#### **-(void)OnError:(NSError \*)aError;**

Called by RPCLib when an error occurs while invoking a web service.

<i>Parameter</i>	<i>Type</i>	<i>Description</i>
aError	NSError	The error. In most cases, this is the error provided by the underlying NSURLConnection. The application code should check the NSError that is sent with the delegate call to see if this was due to a networking or other temporary error. If the error domain is



		<p>“kCFErrorDomainNetwork”, then the code can be interrogated to find out what the networking error was.</p> <p>If the error domain is “RPCLibErrorDomain”, some other issue was the cause. In these cases, the error will be of type <code>RPCLibError</code>, which contains a “userInfo” dictionary with keys of “RPCLibErrorMIMETType” and “RPCLibErrorData” with the server’s error response.</p>
--	--	--

#### **-(void)OnCertificateError:(NSError \*)aError;**

Called by RPCLib when an error occurs while reading the user’s certificate data from the device. This could be an issue with access to the shared keychain, or could be due to the user’s never having executed the SSO app flow once before. The app should then call RPCLayer’s “InvokeSSO” method in response. Handling this callback is mandatory.

<i>Parameter</i>	<i>Type</i>	<i>Description</i>
aError	NSError	The error.

## **3.2 RPCLayer**

The RPCLayer class is the primary point of entry into the RPCLib for client applications. The RPCLayer class provides a single interface for any iOS application to call PwC IdM-protected web services. The RPCLayer’s internals handle the details of certificates, logins, cookies/headers related to authentication, etc and provides a simplified interface to request data from backend systems.

Additionally, RPCLayer provides utility functions to provide access to the individual items stored in the device keychain such as user specific info (GUID, SAPID (aka Employee ID), PPID, SMTOKEN, and COUNTRY). RPCLayer also has simplified methods for simple user authentication and invoking the SSO application, when necessary.

### **3.2.1 Methods**

#### **-(void)RpcCall :(RPCType)aRpcCallType serviceUrl:(NSString \*)aStrEndPoint withData:(RPCObj \*)aRpcObj;**

Method called to invoke a web service. **Note** that the client code should call “SetHandler” method to provide a delegate for events.

<i>Parameter</i>	<i>Type</i>	<i>Description</i>
aRpcCallType	RPCType	One of: RPCSOAP_POST, RPCRestfulJSON_POST, RPCSOAP_GET, RPCRestfulJSON_GET, RPCRestfulJSON_DELETE, RPCRestfulJSON_PUT, RPCRestfulJSON_HEAD
aStrEndPoint	NSString	The service endpoint URL
aRpcObj	RPCObj	The serializable data (or nil)

#### **-(void)SetHandler:(id<IRPCHandler>)aHandler;**

Method called to specify the delegate class that will handle all callback events from RPCLib.

<i>Parameter</i>	<i>Type</i>	<i>Description</i>
------------------	-------------	--------------------

aHandler	<id>IRPCHandler	The delegate class
----------	-----------------	--------------------

**-(PwCUserInfo \*)GetPwCUserInfo;**

Retrieves a PwCUserInfo object which provides a dictionary of all the values stored in PwC specific device keychain.

Parameter	Type	Description
(return)	PwCUserInfo	PwCUserInfo has a property for each of the important pieces of user data: <ul style="list-style-type: none"> <li>• “ppid” - the permanent person ID (PPID) of the user</li> <li>• “guid” - the PwC GUID of the user</li> <li>• “employeeId” - the SAP or employee ID of the user</li> <li>• “country” - the two-letter country code of the user</li> <li>• “email” - the on-premises (non-cloud) email address of the user</li> </ul>

**-(void)AuthUser**

Performs an authentication to a Siteminder endpoint, thus actively verifying the validity of the on-device certificate and the presence of the user as an active status employee in the Firm directory.

**Note** that the client app still needs to call “SetHandler” before making this call. In this case, however, the callback is a success when the “onSingleSignOnSuccess” method is called, and the “OnRPCResponse” delegate method will **not** be called by RPCLib.

Parameter	Type	Description

**-(BOOL)InvokeSSO**

Used by the client app to invoke the SSO app. This is done by invoking the SSO app’s URL protocol, as in “PwCSSO://”.

**Note** that after the SSO app successfully provisions the user, it will attempt to cause the client application to be launched using the client app’s URL protocol. The method will use the first URL scheme in the application’s Info.plist when attempting to determine the URL scheme of the app. This URL scheme can be overridden using the “SetApplicationName” method, but this is not recommended. Apps should be able to handle being launched using a URL protocol and respond accordingly.

**Note** that if this method returns NO, the app should **exit** to prevent an unauthorized user from seeing cached data on screen. The abort() call is a better practice than the exit() call in this case.

Parameter	Type	Description
(return)	BOOL	Returns YES if the SSO app can be found and invoked. Returns NO if the SSO app is not present on the device.

**-(BOOL)isUserProvisioned**

Method to determine if the user has valid certificates and other keychain data. This method makes no network calls, so it executes very quickly and synchronously.

**Note** that this method should be used every time the app launches or moves from background to foreground. If the result is NO, the client app **must** call the “InvokeSSO” method.

Parameter	Type	Description
(return)	BOOL	Returns YES if the user’s data appears correctly in the shared keychain. Returns NO if there is an issue with the user’s keychain data..

**-(void)logout**

Method that removes any session-level tokens from the application's shared storage space. **Note** that this is mandatory for DC2 apps.

<i>Parameter</i>	<i>Type</i>	<i>Description</i>

#### **-(void)SetRetryAttempt:(int)aIntNoOfRetryAttempts**

Method that sets the number of times the library will attempt to start a connection to the service endpoint.

<i>Parameter</i>	<i>Type</i>	<i>Description</i>
aIntNoOfRetryAttempts	int	The number of retry attempts to make before the web service call is considered failed. If no value is specified, the value of 3 will be used. The normal value of this is between 3 and 5. Setting this too high may cause delays in finding out that the network is unavailable.

#### **-(NSMutableArray \*)setHeaderDictionary**

#### **:(NSMutableDictionary \*)additionalHeaders**

Allows client apps to send additional HTTP headers with the web service request.

<i>Parameter</i>	<i>Type</i>	<i>Description</i>
additionalHeaders	NSMutableDictionary	Additional headers to be sent with the web service call. For each entry, the dictionary key will be used as the header name, and the value will be used as the header value.

## **4 RPCLib Use Cases**

The following sections will attempt to lay out which RPCLib calls to include in your code, given your app's situation. In general, apps fall into two main categories:

- Apps that have API layers that are protected by Siteminder or IdAM
- Apps that obtain data in some other manner, but need to use the RPCLib for user authentication only

These two use cases are broken out below, with their respective RPCLib calls.

### ***4.1 Authentication and Accessing Web Service Data***

If the app will need access to Siteminder- or IdAM-protected data, the RPCLib takes over much of the responsibility for networking code. The app interacts with a simple abstraction that hides authentication, session expiration, and other details. To take advantage of this, in addition to the **mandatory** "Application Launch" steps, the application code should do the following:

1. If passing data, as in an HTTP POST, create an object that implements the IRPObject protocol.
2. Create an object that implements the IRPCHandler delegate calls.
3. Allocate and init the RPCLayer object.
4. In a thread other than the main or UI thread, call RPCCall to start the request.
5. Handle the responses within your custom IRPCHandler.

Some tips:

- Implement the IRPCHandler as an NSOperation that starts the RPCCall when the NSOperation starts.
- Keep a strong reference to the RPCLayer object. This can be done by making an instance of RPCLayer a property of your client class.
- It is always best practice to execute networking operations on a thread other than the main thread of execution. When the delegate calls are made, they are made on the same thread of execution that initiated the request. It is often the case that the resulting logic should be performed on the main thread, if it includes UI updates.

The following example code demonstrates a web service call:

```
NSString
*strURL=@"http://mpeopleloc-stg.pwcinternal.com/peoplefinder/pf/services/searchUser.
do";

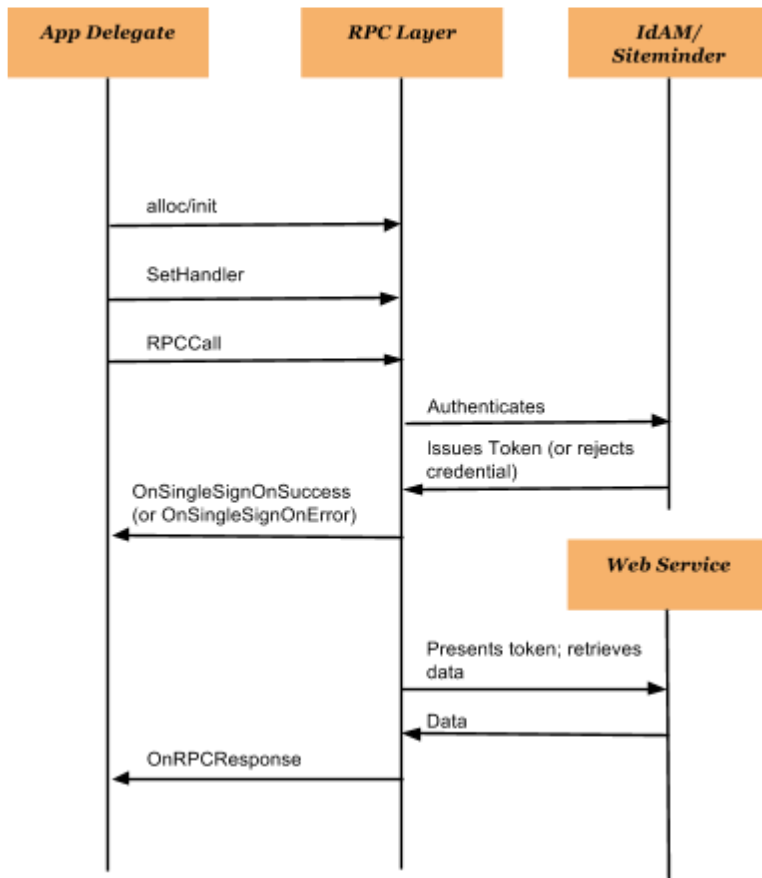
/**
 * This search service accepts a POSTed JSON message in the following format:
 * {
 *   "firstName" : "B",
 *   "lastName"  : "Moritz"
 * }
 */
NSMutableDictionary *srchInputD = [[NSMutableDictionary alloc] init];
[srchInputD setValue:@"Smith" forKey:@"lastName"];
RPCObj *postObj = [[RPCObj alloc] init];
[postObj setIDictInputDictionary:srchInputD];
// "rpclayer" is a strongly-referenced @property
[self.rpclayer SetHandler:self];

dispatch_async(dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT,
                                         (unsigned long)NULL), ^(void) {

    // Calling "RpcCall" with "restful JSON POST" and the search arguments
    [self.rpclayer RpcCall:RPCRestfulJSON_POST
                     serviceUrl:strURL
                   withData: postObj];

});
```

This is the full interaction, presented pictorially:



### 4.1.1 Cordova Projects

For Cordova projects, the RPCLib will appear as a window-level variable named “PwCRPC”. Here is an example of an HTTP GET call:

```

window.PwCRPC.callJSONService(
    function(data) { // success callback function, accepts new data
        console.log('callJSONService SUCCESS ' + JSON.stringify(data));
    },
    function(err) { // error callback function
        console.log('callJSONService ERROR' + JSON.stringify(err));
    },
    getUrl           // service endpoint URL
);
  
```

In the case of PUT or POST requests, there are two extra arguments:

```

var postData;
postData.firstName = "B";
postData.lastName = "Moritz";
window.PwCRPC.callJSONService(
    function(data) { // success callback function, accepts new data
        console.log('callJSONService SUCCESS ' + JSON.stringify(data));
    },
    function(err) { // error callback function
        console.log('callJSONService ERROR' + JSON.stringify(err));
    },
    postData,       // service endpoint URL
    postData,       // JS object to PUT or POST
    "POST"          // HTTPS Verb: One of "PUT" or "POST".
);
  
```

```

// "GET" is inferred by not passing in a data object
);

```

## 4.2 Authentication Only

In addition to the **mandatory** “Application Launch” steps (below), the application code should do the following:

1. Create an object that implements the IRPCHandler delegate calls.
2. Create the RPCLayer object.
3. In a thread other than the main or UI thread, call “AuthUser” to start the request.
4. Handle the responses within the custom IRPCHandler.
  - a. **Remember** that the method for a successful authentication-only “AuthUser” call is the “OnSingleSignOnSuccess” method. “OnRPCResponse” will not be called.

## 4.3 Application Lifecycle Events

### 4.3.1 Application Launch (mandatory)

The following steps must be added after the application launches and when it returns from the background:

1. Check to see if the user’s device is provisioned properly using the “IsUserProvisioned” method.
2. If the user is not provisioned, attempt to call the SSO application by using the “InvokeSSO” call. Note that this is a synchronous call that does not use the delegate protocol.
3. If the InvokeSSO method returns NO, the SSO app is not present on the device, and there are no other steps that can be taken. At this point, the app should display a message to the user notifying him of the error.
  - a. When the user acknowledges the alert, the app should **close** to prevent users from seeing any cached data that may be accidentally shown.

This approach ensures that users do not see data for which they are not authorized, and also provides a nice user experience if users have forgotten to provision their devices for use with PwC apps.

The following code demonstrates the mandatory items:

```

- (BOOL)application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
    // UI initialization code ...
    //

    RPCLayer *rpc=[[RPCLayer alloc]init];
    if (![rpc isUserProvisioned]) {
        if (![rpc InvokeSSO]){
            UIAlertController* alert =
                [UIAlertController alertControllerWithTitle:@"Secure Sign-On App Not Found"
                 message:@"Please install the 'Secure SSO' application then try again"
                 preferredStyle:UIAlertControllerStyleAlert];

            UIAlertAction* defaultAction = [UIAlertAction actionWithTitle:@"OK"
                style:UIAlertActionStyleDefault
                handler:^(UIAlertAction * action) {
                    abort();
                }];

            [alert addAction:defaultAction];
            NSArray *appWindows = [UIApplication sharedApplication].windows;

```

```

        for(UIWindow *window in appWindows){
            [window.rootViewController presentViewController:alert animated:YES
completion:nil];
        }
        return NO;
    }
}
return YES;
}

```

### 4.3.1.1 Cordova Projects

The same mandatory requirements are present in Cordova projects. The method names are slightly different though. Here is an equivalent Cordova implementation:

```

window.PwCRPC.getCurrentUserData(
    function(user) { // success function
        console.log('logged in user is ' + JSON.stringify(user));
    },
    function(err) { // user is not provisioned
        console.log('user is not provisioned ERROR' + err);
        window.PwCRPC.invokeSSO( // SSO app will launch now
            function() {
                console.log('SSO found');
            },
            function() {
                console.log('Not able to open the SSO app');
            }
        );
    }
);

```

### 4.3.2 Application Leaves Foreground (mandatory for DC2)

Applications that are rated DC2 must have affirmative logout mechanisms that tear down server sessions so that we limit the exposure of temporary tokens. It is an uncommon user experience to be presented with a “logout” button in a mobile app. Instead, we listen for the application to be sent to the background as a signal that the user is no longer actively using the app, and perform a logout at that point. To accomplish this, the following steps must be added to the logic that executes when the app is sent to the background:

1. Create an RPCLayer object
2. Invoke the “logout” method
  - a. This will remove all client side cookies
  - b. RPC lib then starts an HTTP GET request to each URL listed in the RPCConfiguration file’s “AuthLogoutURLs” array of String values. See the configuration section on details for the correct settings

The following code demonstrates the mandatory items:

```

- (void)applicationDidEnterBackground:(UIApplication *)application
{
    RPCLayer *rpc=[[RPCLayer alloc]init];
    if ([rpc isUserProvisioned]) {
        [rpc logout];
    }
}

```

```
}
```

The “logout” method is available under Cordova, as well.

## ***5 Integrating the RPCLib into a Project***

### ***5.1 IDE Activities***

Follow the process below to integrate the RPC Library with a mobile application. This process assumes you have downloaded the zipped binary distribution of the RPCLib and that the device is provisioned on the PwC network.

1. Open your application project
2. Create a new group folder in your project space with name “RPCLib”.
3. Drag the folders from the unzipped binary distribution into the “RPCLib” group. This will contain, among other items:
  - 3.1. The device and simulator static libraries
  - 3.2. The current header files (See RPCLayer.h for version information)
  - 3.3. RPCCConfigurations.plist
  - 3.4. updateRPCCConfig.sh
  - 3.5. Cordova plugin related files in subdirectories named “cordova” and “Plugin”. For native applications, these are not needed and should be deleted.
4. Import the following libraries into the project under “Build Phases->Link Binary with Libraries”
  - 1.1. libxml2.tbd (or .dylib)
  - 1.2. libz.tbd (or .dylib)
  - 1.3. Security.framework (if not already present in the project)
  - 1.4. SystemConfiguration.framework
5. Follow the instructions in the Mobile App Dev Standards to create xcode schemes to match the “Dev”, “Stage” and “Prod” environments.
6. In your target’s “Build Phases” settings, create a new build phase by clicking on the “+” icon:
  - 6.1. Choose new “Run Script phase” from the plus sign dropdown
  - 6.2. Rename the phase “Update RPC Config”
  - 6.3. Paste the contents of the “updateRPCCConfig.sh” script into the script contents window
  - 6.4. Drag this phase to reorder it just below the “Target Dependencies” phase.
7. Define a protocol handling URL under your project’s “Info” tab under “URL Types”, if your project doesn’t already have one. Be sure to name the protocol URL prefixed with “PwC” in



order to reduce the chance of a protocol name clash with other apps installed on the users' devices.

8. Under project “Capabilities”, under “Keychain Sharing”, add the following keychain access group: “com.pwc.us.mobility.PwCKeychainAccessGrp”. Xcode may display errors that the app ID does not have keychain sharing specified, but these can be ignored.
9. Follow the coding instructions in the next section to make RPCLib calls to authenticate and/or access PwC webservice data.
10. Build your XCode application

## 5.2 Configuration

The configuration settings that are necessary for the RPCLib to work correctly are contained within the RPCLibConfiguration.plist settings file that is part of the binary distribution. By default, the file is shipped with settings that will work for Siteminder-protected services or apps with only “Authentication” use case requirements, described in “RPCLib Use Cases” that service Siteminder-participating territories. The following table should help with configuration:

Setting	Description
AuthURL	<p>For apps that have “Authentication only” use cases, this URL provides certificate-based authentication when calling the “AuthUser” method. This setting can be any URL that complies with the following requirements:</p> <ul style="list-style-type: none"> <li>• The URL should perform certificate-based authentication. Siteminder or IdAM normally comply if set up in “cert” or “cert or forms” configurations.</li> <li>• The URL should accept an HTTP GET request</li> <li>• The URL should return an HTTP 200 in the case of a successful authentication request</li> <li>• The URL should respond with a MIME type of “application/json”.</li> </ul> <p>The distribution ships with valid values for Siteminder in the AuthURL settings. For IdAM, a URL should be set up within the app’s backend services that is protected by IdAM and will return JSON data (even empty data) as a result of a successful authentication flow.</p> <p>For “Services” use cases, this setting is only used if the service endpoint returns a 401 http error code. In that case, this should be the URL of a service that will perform certificate-based authentication and provide an HTTP cookie with a value that will allow subsequent calls to the application backend’s service endpoints.</p> <p>There is an “AuthURL” defined for each environment, under the keys “Dev”, “Stage”, and “Prod”. This is dependent on following the setup steps completely.</p>
AuthLogoutURLs	<p>Affirmative logouts must be performed for DC2 and above applications. When an app is put into the background, these URLs are called to remove session-side sessions and invalidate temporary tokens.</p> <p>For Siteminder, these are already set correctly in the file that ships with the distribution. For IdAM, use the URLs in the “<a href="#">IdAM Application Integration Guide</a>” in Spark.</p>
DefaultConnectionTimeout	<p>Number, in seconds, to wait for a server response before the connection is stopped by the operating system. Since all connections are retried, this may be not a strict elapsed time.</p>

## 5.3 Cordova Integration

RPCLib can be used in HTML hybrid applications that use the Cordova framework. The RPCLib is available as a Cordova plugin from an internally-hosted repository. To integrate the RPCLib into your Cordova project:

1. Obtain a binary distribution of the RPCLib
2. Copy the RPCConfiguration.plist file from the binary distribution into your project's root directory.
3. Make any adjustments necessary for the RPCConfiguration.plist file, then check it into the source code repository as a permanent part of your project.
4. Add the RPCLib as a Cordova plugin. This depends on your project structure:
  - a. If you are using Ionic Framework: Add RPCLib's location to your Cordova project's "package.json" file under "cordovaPlugins". The value to use is "git://usatlkapple003.pwcinternal.com/com.pwc.us.mobility.pwcrpc.git"
  - b. If you are using Cordova alone, use the "plugin add" command to add RPCLib as a plugin:  

```
cordova plugin add
git://usatlkapple003.pwcinternal.com/com.pwc.us.mobility.pwcrpc.git
```

## 6 RPCLib Distribution

### 6.1 Binary Distribution

The RPC library binaries can be retrieved from the [PwC IT SharePoint distribution site](#).

<https://teamspaceint-deg.pwcinternal.com/sites/5f89537e754e00e3a697>

Access to the distribution can be requested from the Global Service Desk.

### 6.2 Sample App Distribution

A sample application has been provided to illustrate the usage of RPCLib. It is available on the following [PwC IT SharePoint distribution site](#).

<https://teamspaceint-deg.pwcinternal.com/sites/5f89537e754e00e3a697>

Access to the distribution can be requested from the Global Service Desk.

## 7 Appendix

### 7.1 Keychain Items

As part of the SSO app process, certificates and other data is written to the device keychain. The following table lists the items written to the keychain and the metadata that is written with them.

Item	Description	Settings
Identity Certificate	The user-specific X.509 certificate. This consists of a certificate and a private key.	Type: SecIdentity (kSecClassIdentity) Label: (GUID of user) Summary: (GUID of user)

Guid	The user's GUID.	Type: Generic Password Account: UserGUID Service: SSOService
PPID	The user's PPID.	Type: Generic Password Account: UserPPID Service: SSOService
Employee ID	The user's Employee ID. For some territories, this is not available and will not be written.	Type: Generic Password Account: UserSAPID Service: SSOService
Email	The user's on-prem email address.	Type: Generic Password Account: UserEMAIL Service: SSOService
Country	The user's two-letter country code, as listed in the Enterprise Directory.	Type: Generic Password Account: SMHdCountryVal Service: SSOService
Root Certificate	SHA-1 root certificate. Beginning in mid 2016, this will be both the SHA-1 root and the SHA-2 root certificates.	Type: Certificate (kSecClassCertificate) Summary: "PricewaterhouseCoopers Root" Label: "PricewaterhouseCoopers Root" Issuer: "PricewaterhouseCoopers Root"
"Policy" Intermediate Certificate	SHA-1	Type: Certificate (kSecClassCertificate) Summary: "PricewaterhouseCoopers Policy" Label: "PricewaterhouseCoopers Policy" Issuer: "PricewaterhouseCoopers Root"
"Issuing1" Intermediate Certificate	SHA-1	Type: Certificate (kSecClassCertificate) Summary: "PricewaterhouseCoopers Issuing1" Label: "PricewaterhouseCoopers Issuing1" Issuer: "PricewaterhouseCoopers Policy"
"Issuing2" Intermediate Certificate	SHA-1	Type: Certificate (kSecClassCertificate) Summary: "PricewaterhouseCoopers Issuing2" Label: "PricewaterhouseCoopers Issuing2" Issuer: "PricewaterhouseCoopers Policy"

## 7.2 Error Codes

**OnError** is the delegate method which gets called when any error occurs while calling the web service. It can be called under following cases:

1. NSURLConnection – **didFailWithError: (NSError \*) error**

Some of the common error codes under CFNetworkErrors are

```
*kCFHostErrorHostNotFound = 1
*kCFHostErrorUnknown = 2
kCFSOCKSErrorUnknownClientVersion = 100
kCFSOCKSErrorUnsupportedServerVersion = 101
kCFSOCKS4ErrorRequestFailed = 110
kCFSOCKS4ErrorIdentdFailed = 111
kCFSOCKS4ErrorIdConflict = 112
kCFSOCKS4ErrorUnknownStatusCode = 113
kCFSOCKS5ErrorBadState = 120
kCFSOCKS5ErrorBadResponseAddr = 121
kCFSOCKS5ErrorBadCredentials = 122
kCFSOCKS5ErrorUnsupportedNegotiationMethod = 123
kCFSOCKS5ErrorNoAcceptableMethod = 124
kCFFTPErrorUnexpectedStatusCode = 200
kCFErrorHTTPAuthenticationTypeUnsupported = 300
kCFErrorHTTPBadCredentials = 301
kCFErrorHTTPConnectionLost = 302
kCFErrorHTTPParseFailure = 303
kCFErrorHTTPRedirectionLoopDetected = 304
kCFErrorHTTPBadURL = 305
kCFErrorHTTPProxyConnectionFailure = 306
kCFErrorHTTPBadProxyCredentials = 307
kCFErrorPACFileError = 308
kCFErrorPACFileAuth = 309
kCFErrorHTTPSPProxyConnectionFailure = 310
* kCFURLErrorUnknown = -998
kCFURLErrorCancelled = -999
kCFURLErrorBadURL = -1000
*kCFURLErrorTimedOut = -1001
kCFURLErrorUnsupportedURL = -1002
*kCFURLErrorCannotFindHost = -1003
*kCFURLErrorCannotConnectToHost = -1004
kCFURLErrorNetworkConnectionLost = -1005
kCFURLErrorDNSLookupFailed = -1006
kCFURLErrorHTTPTooManyRedirects = -1007
kCFURLErrorResourceUnavailable = -1008
*kCFURLErrorNotConnectedToInternet = -1009
kCFURLErrorRedirectToNonExistentLocation = -1010
kCFURLErrorBadServerResponse = -1011
kCFURLErrorUserCancelledAuthentication = -1012
kCFURLErrorUserAuthenticationRequired = -1013
kCFURLErrorZeroByteResource = -1014
kCFURLErrorCannotDecodeRawData = -1015
kCFURLErrorCannotDecodeContentData = -1016
kCFURLErrorCannotParseResponse = -1017
kCFURLErrorInternationalRoamingOff = -1018
kCFURLErrorCallIsActive = -1019
kCFURLErrorDataNotAllowed = -1020
kCFURLErrorRequestBodyStreamExhausted = -1021
kCFURLErrorFileDoesNotExist = -1100
kCFURLErrorFileIsDirectory = -1101
kCFURLErrorNoPermissionsToReadFile = -1102
kCFURLErrorDataLengthExceedsMaximum = -1103
kCFURLErrorSecureConnectionFailed = -1200
kCFURLErrorServerCertificateHasBadDate = -1201
kCFURLErrorServerCertificateUntrusted = -1202
kCFURLErrorServerCertificateHasUnknownRoot = -1203
```

```

kCFURLErrorServerCertificateNotYetValid = -1204
kCFURLErrorClientCertificateRejected = -1205
kCFURLErrorClientCertificateRequired = -1206
kCFURLErrorCannotLoadFromNetwork = -2000
kCFURLErrorCannotCreateFile = -3000
kCFURLErrorCannotOpenFile = -3001
kCFURLErrorCannotCloseFile = -3002
kCFURLErrorCannotWriteToFile = -3003
kCFURLErrorCannotRemoveFile = -3004
kCFURLErrorCannotMoveFile = -3005
kCFURLErrorDownloadDecodingFailedMidStream = -3006
kCFURLErrorDownloadDecodingFailedToComplete = -3007
kCFHTTPCookieCannotParseCookieFile = -4000
kCFNetServiceErrorUnknown = -72000L
kCFNetServiceErrorCollision = -72001L
kCFNetServiceErrorNotFound = -72002L
kCFNetServiceErrorInProgress = -72003L
kCFNetServiceErrorBadArgument = -72004L
kCFNetServiceErrorCancel = -72005L
kCFNetServiceErrorInvalid = -72006L
kCFNetServiceErrorTimeout = -72007L
kCFNetServiceErrorDNSServiceFailure = -73000L

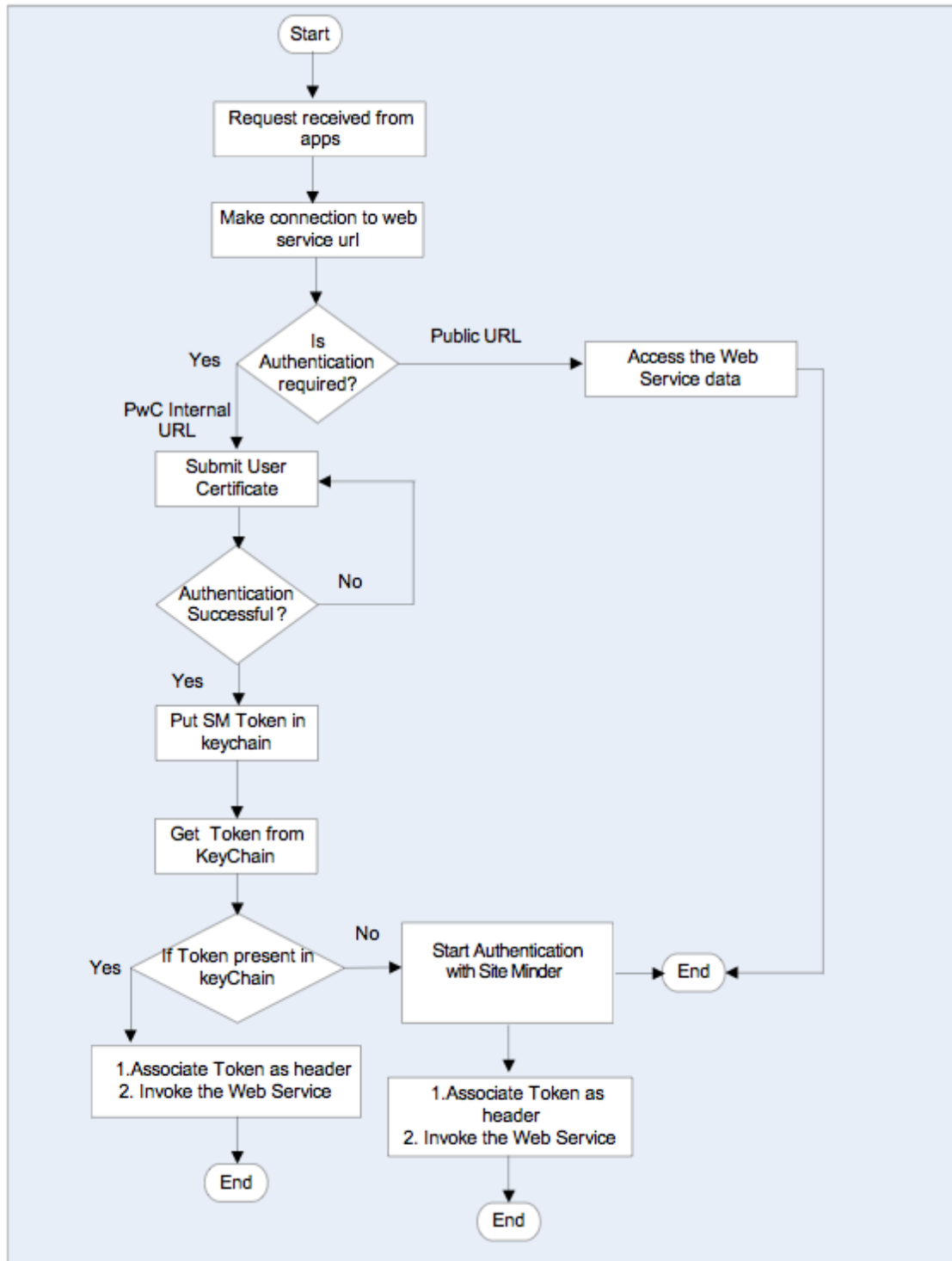
```

- \* - **For these specific error codes, RPCLib will perform retry attempts until the number of retries reaches the limit set in SetRetryAttempt.** If the connection still fails, then the error is returned.
- \* If the web or app server responds with a 400 or 500 style error code (http status code 4xx or 5xx), RPCLib will invoke the OnError callback with the NSError dictionary, but with additional data. This may be not set if the server does not send any additional data. In the NSError's "userInfo" dictionary, two keys with values will be set:
  - \* RPCLibErrorData - The actual text that the server returned, as type NSData.
  - \* RPCLibErrorMIMETYPE - The mime type that the server returned when it sent the error. This can be used with the error data itself to extract messages, JSON data, etc, if the server sends such data.

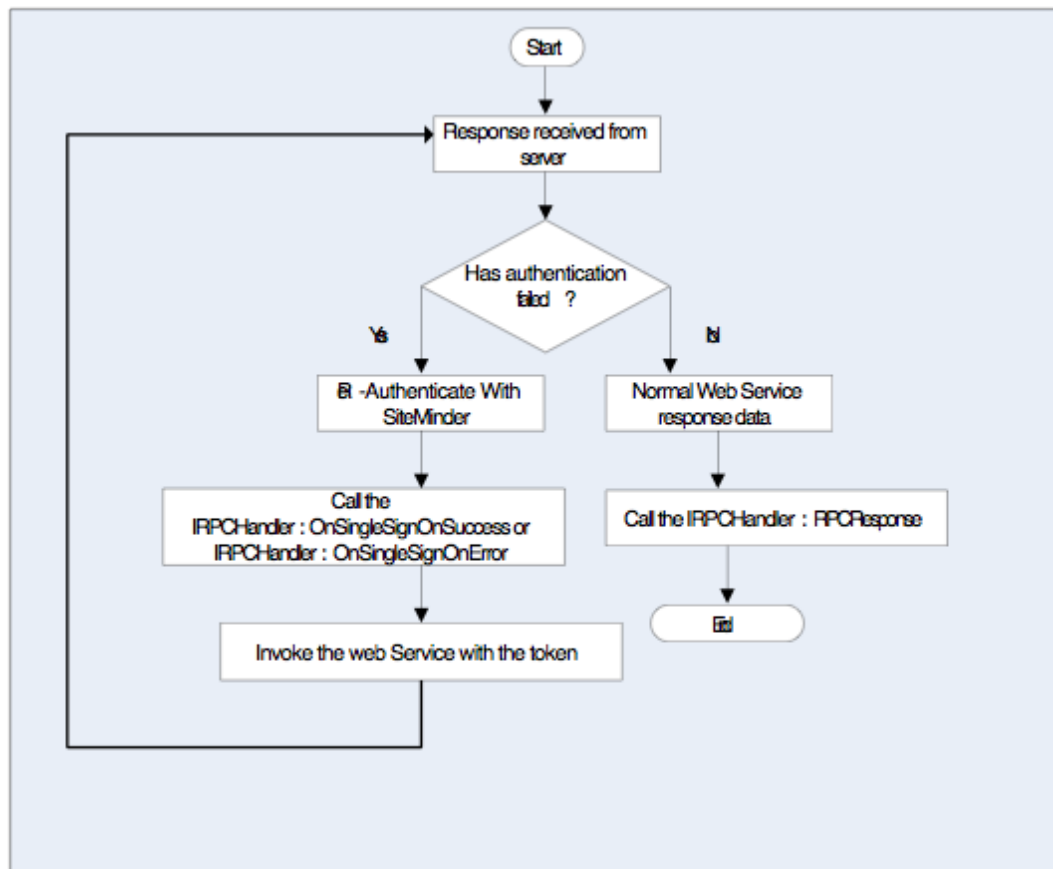
## 7.3 Detailed Internal Flow

The internal working of RPCLayer is depicted as below:

### Request Handling Flow:



### Response Handling Flow:



## **8 Revision History**

<b>Revision Date</b>	<b>Description</b>	<b>Author</b>
3.3 / February 2016	<ul style="list-style-type: none"> <li>Revised for version 3.3 functionality</li> <li>Added configuration settings section</li> </ul>	Tom Sanidas
3.2.2 (renumbered) / July 2015	<ul style="list-style-type: none"> <li>Renumbered to coincide with RPCLib version</li> <li>Introduced API and Use Case sections</li> </ul>	Tom Sanidas
3.5/25-Aug-2014	<ul style="list-style-type: none"> <li>Changes to describe version 3.2: InvokeSSO change, IsUserProvisioned change and error code changes in Appendix; Other cleanup items.</li> </ul>	Tom Sanidas
3.4/25-Mar-2014	<ul style="list-style-type: none"> <li>Changes to describe version 3.1.3</li> </ul>	Tom Sanidas
3.3/11-Oct-2013	<ul style="list-style-type: none"> <li>Added changes related to v3.1.0 and v3.1.1</li> </ul>	Padman Balasubramanian
3.2/02 Jul 2013	<ul style="list-style-type: none"> <li>SetTargetURL unsupported in v3.0.0</li> </ul>	Arun Kumar

