React Native Integration

Following document will highlight the integration of Rel-ID with React Native Reference application and how one can get the developer environment up on their machine.

# Setup

* React Native Installation
  + Follow the link below to install react native environment for building the reference app code. Complete the installation for your target platform and OS.
    - <https://facebook.github.io/react-native/docs/getting-started.html>
* Github account creation and checkout of the Rel-ID React Native Reference App code.
  + Create a github account from the following link and get access to the github repository.
    - <https://github.com/join?source=header-home>
  + Install SourceTree or GitHub client to pull the repositories from the remote location.
    - <https://www.sourcetreeapp.com/>
    - [https://desktop.github.com](https://desktop.github.com/)
  + Checkout the code from the remote location from the below repository link using your credentials to login to the client.
    - <https://github.com/Team-Uniken/REL-ID_internal.git>
* If using Mac, it will be advised to use Xcode or nuclide as the editor. If using Windows machine, use either Android Studio or Atom for editing React native code.

# Rel-ID Integration Specification

REL-ID is a digital trust platform that connects things, securely. It creates a closed, private, massively scalable, networked application ecosystem to protect enterprise applications and data from unauthorized and fraudulent access and tampering.

The REL-ID API-SDK provides the following features that enable applications to leapfrog ahead in terms of securing themselves - mutual identity and authentication, device fingerprinting and binding, privacy of data, and the digital network adapter (aka DNA).

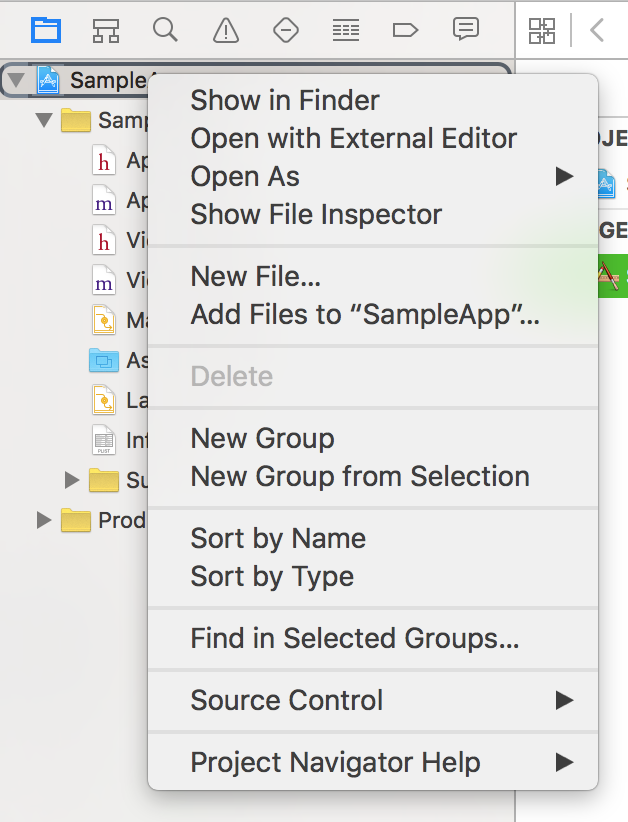
Post checkout, place the following Rel-ID core files in the following locations in the code for the respective platforms.

## **iOS**

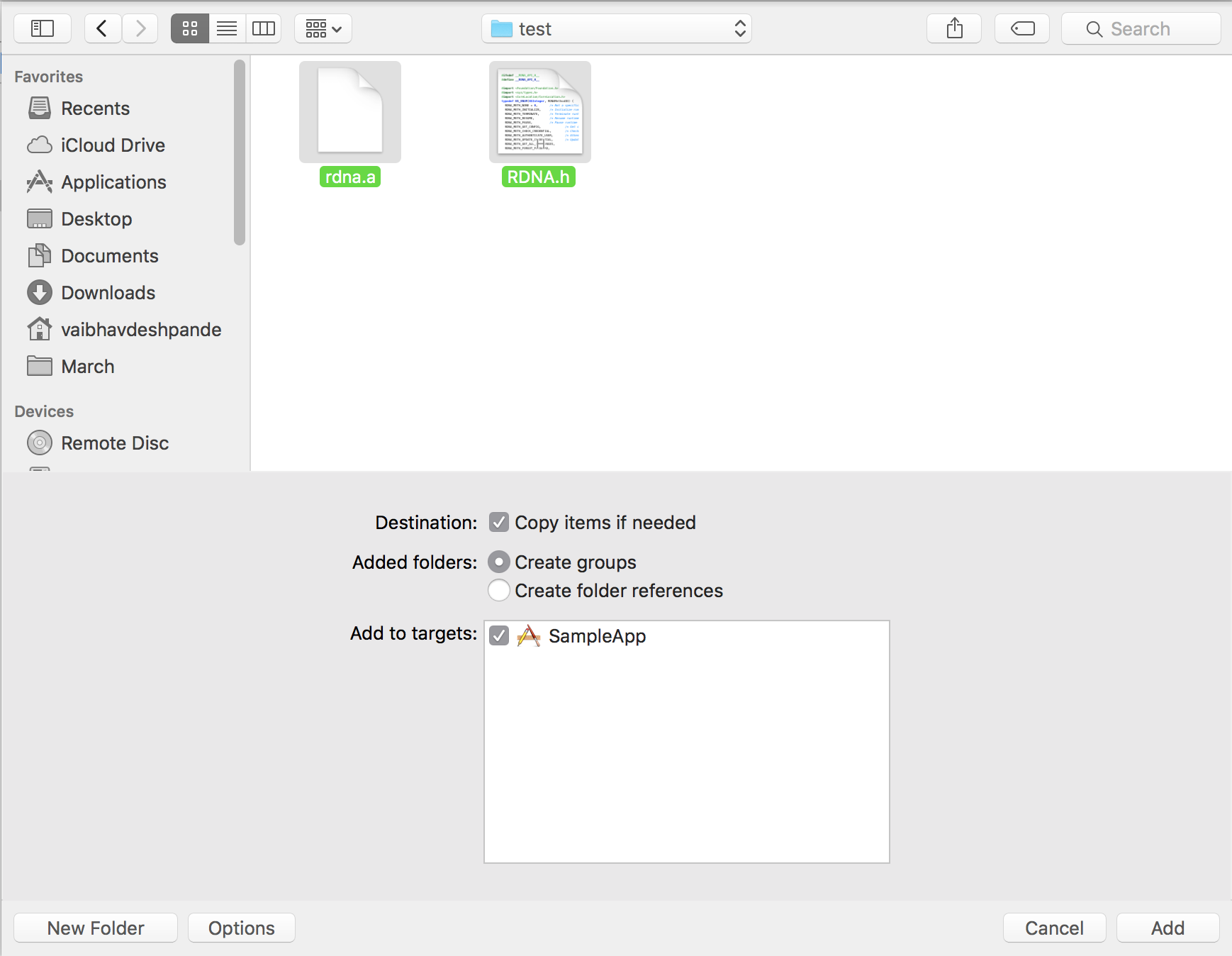
**Adding the SDK to your Xcode project**

If you don't already have the REL-ID API-SDK, grab and unzip it.

#### **Add the RDNA.h and RDNA.a files to the project.**



Right-click on the checkout project from the steps mentioned earlier. Let’s call it “**SampleApp**” and choose **Add Files To "SampleApp"** asshown above**.**



Add the rdna.a and RDNA.h.

#### **Add other frameworks that the SDK requires**

The SDK depends on the following iOS development frameworks which may not already be part of your project:

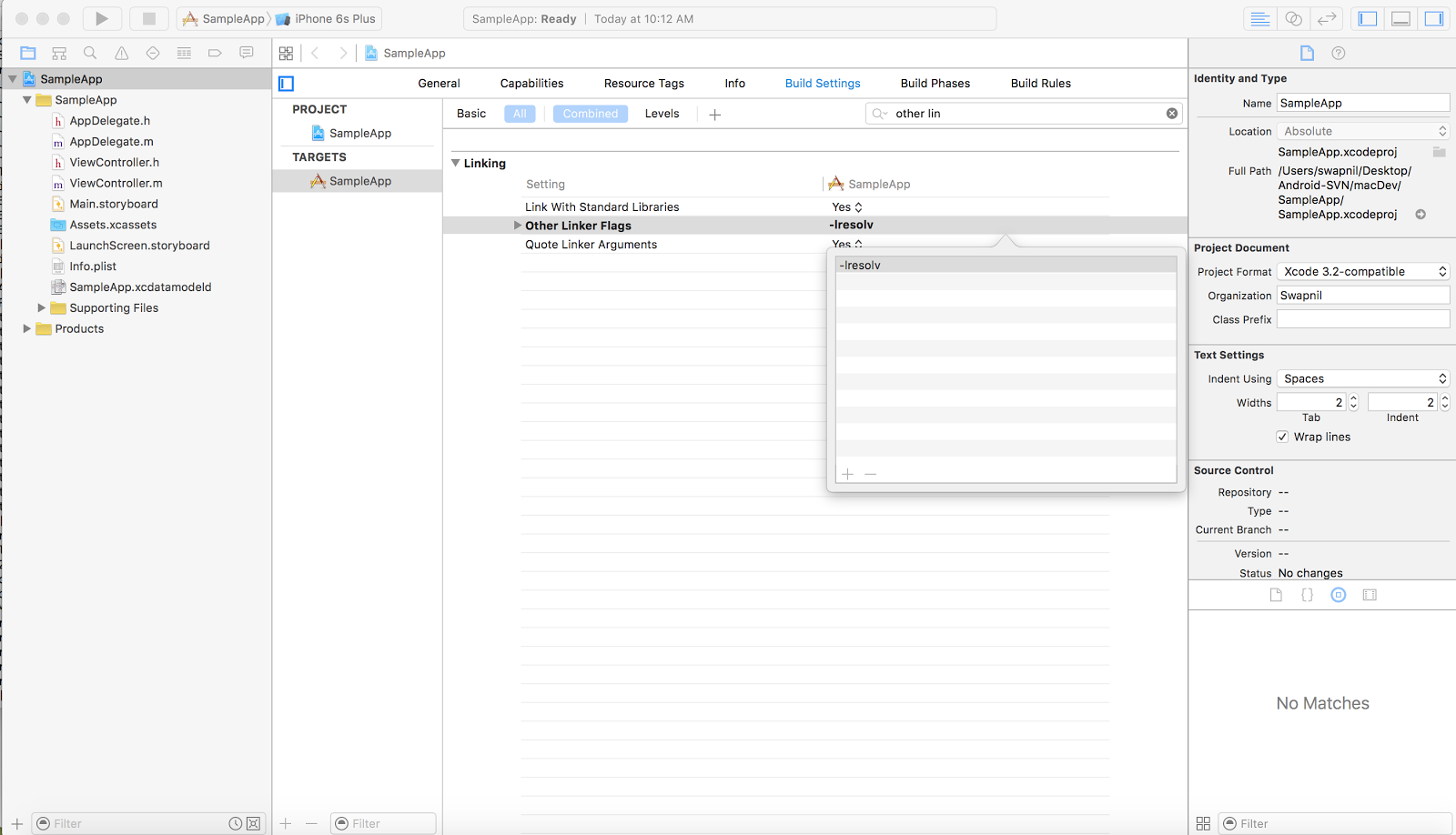
* AdSupport
* CoreTelephony
* CoreLocation

Once the SDK is referenced somewhere in your app, it automatically links these frameworks.

The quickest way to add a reference is to open up ViewController.m for an Objective-C project, import the library.

#### **Adding the "-lresolv" Linker Flag**

1. Select the project file from the project navigator on the far left side of the window.
2. Select the target for where you want to add the linker flag.
3. Select the "Build Settings" tab
4. Choose "All" to show all Build Settings.
5. Scroll down to the "Linking" section, and double-click to the right of where it says "Other Linking Flags".
6. A box will appear, Click on the "+" button to add a new linker flag.
7. Type "-lresolv" (no quotes) and press enter.



#### Rebuild your project

Rebuild and run your project. You'll still see a white screen, but now you'll see a log in the Xcode console indicating what version of the REL-ID API-SDK you have.

## **Android**

If you don't already have the REL-ID API-SDK, grab and unzip it.

* Add in application level build.grade file, i.e. at ../SampleApp/android/build.gradle path, add the highlighted lines of code in yellow accordingly:

allprojects {

repositories {

mavenLocal()

jcenter()

maven {

// All of React Native (JS, Obj-C sources, Android binaries) is installed from npm

url "$projectDir/../../node\_modules/react-native/android"

}

flatDir {

dirs 'libs'

}

}

}

* In module level build.gradle, i.e. at ../SampleApp/android/App/build.gradle path, add the following code to add the SDK file in :

dependencies {

compile fileTree(dir: "libs", include: ["\*.jar"])

compile "com.android.support:appcompat-v7:23.0.1"

compile "com.facebook.react:react-native:+" // From node\_modules

compile(name:'app-sdk-debug', ext:'aar')

compile project(':ExtraDimensions')

}

* Place the 'app-sdk-debug.aar' file from the API SDK zip and place it inside a new folder named ‘libs’ at path “../SampleApp/android/app/”.

# QBank Web Application Integration

QBank is a web application which showcases basic banking features that can be exploited on mobile. It provides APIs to fetch account details, account balances, etc. Follow the below steps to setup QBank application:

1. Copy uniken-QBApp-deployable.tgz to the folder of your choice on Linux machine.
2. Run following command as non-root user.
   1. tar -zcvf uniken-QBApp-deployable.tgz
   2. . ./ init-vpsi-paths.sh
   3. ./start-all.sh
3. After you have run the above steps, the QBank Web application will be listening on port 3080 and PostgreSQL will be listening on port 5432.

# API SDK Integration

This section will walk you through the API integration of the basic APIs through which one can enable Rel-ID in your application. This will contain processes like Initialization of Rel-ID, Primary Activation, Secondary Device Activation, Normal login, etc.

## Initialize

Initialization is the process of setting up the connection of the client device to the server. Below is the code snippet for initialization:

###### Request Code Snippet:

ReactRdna.initialize(ReactRdna.agentInfo,ReactRdna.GatewayHost,ReactRdna.GatewayPort,ReactRdna.RdnaCipherSpecs,ReactRdna.RdnaCipherSalt,jsonProxySettings,(response) => {

if (response) {

console.log('immediate response is'+response[0].error);

…

}else{

console.log('immediate response is'+response[0].error);

…

}

});

###### Response Code Snippet:

DeviceEventEmitter.addListener('onInitializeCompleted', function (e) {

console.log('immediate response is'+e.response);

responseJson = JSON.parse(e.response);

if(responseJson.errCode == 0){

// Successfully initialized

} else {

// Some error has happened. SDK was not initialized.

}

});

## CheckChallenge

As part of the API call sequence to authenticate an end-user, the API-client receives challenges in the form of challengeJson json object. The API-client would then receive the response to the challenge from the end-user, and use the checkChallenge API to pass the challenge responses to the server. The response will be validated by the server. The server would then process the response and the API-client will receive a response object representing the result of processing the response for the challenge. The server response would indicate that whether the response was validated successfully or not. This API should only be called to authenticate an end-user, and post authentication this API should not be used.

###### Request Code Snippet:

ReactRdna.checkChallenges(JSON.stringify(challengeJson), value, (response) => {

if (response[0].error == 0) {

console.log('immediate response is' + response[0].error);

// alert(response[0].error);

} else {

console.log('immediate response is' + response[0].error);

alert(response[0].error);

}

});

###### Response Code Snippet:

let subscriptions = DeviceEventEmitter.addListener(

'onCheckChallengeResponseStatus',

this.onCheckChallengeResponseStatus.bind(this)

);

## GetRegisteredDeviceDetails

This API fetches the list of devices registered to the user whose userID/username is provided as input parameter.

###### Request Code Snippet:

ReactRdna.getRegisteredDeviceDetails(value, (response) => {

…

});

###### Response Code Snippet:

onGetDevice = DeviceEventEmitter.addListener('onGetRegistredDeviceDetails', this.onGetRegistredDeviceDetails.bind(this));

## UpdateDeviceDetails

UpdateDeviceDetails API is to be used by the API-client to persist any changes made to the list of registered devices for that user.

###### Request Code Snippet:

ReactRdna.updateDeviceDetails(value, JSON.stringify(devicesList), (response) => {

…

});

###### Response Code Snippet:

onDeviceUp = DeviceEventEmitter.addListener('onUpdateDeviceDetails', this.onUpdateDeviceDetails.bind(this));

## Pause and Resume

The pause and resume routines make it possible to persist the in-session state of the API runtime and restore the runtime from the previously persisted state.

This is useful in case of limited configuration devices and platforms - such as smartphone device platforms like Android, iOS and WindowsPhone. In these platforms, a running application could be swapped out of memory due to 'crowding’ by other running applications, only to be swapped back in when the user chooses to access that application again.

### PauseRuntime API

###### Request Code Snippet:

ReactRdna.pauseRuntime((response) => {

if (response) {

if (response[0].error == 0) {

AsyncStorage.setItem("savedContext", response[0].response);

}

console.log('immediate response is' + response[0].error);

} else {

console.log('immediate response is' + response[0].error);

}

})

###### Response Code Snippet:

var onPauseCompletedListener = DeviceEventEmitter.addListener('onPauseCompleted', function(e) {

console.log('On Pause Completed:');

console.log('immediate response is' + e.response);

responseJson = JSON.parse(e.response);

if (responseJson.errCode == 0) {

console.log('Pause Successfull');

} else {

alert('Failed to Pause with Error ' + responseJson.errCode);

}

});

### ResumeRuntime API

###### Request Code Snippet:

ReactRdna.resumeRuntime(value, jsonProxySettings, (response) => {

if (response) {

console.log('immediate response is' + response[0].error);

} else {

console.log('immediate response is' + response[0].error);

}

})

###### Response Code Snippet:

var onResumeCompletedListener = DeviceEventEmitter.addListener('onResumeCompleted', function(e) {

console.log('On Resume Completed:');

console.log('immediate response is' + e.response);

responseJson = JSON.parse(e.response);

if (responseJson.errCode == 0) {

console.log('Resume Successfull');

AsyncStorage.setItem("savedContext", "");

} else {

AsyncStorage.setItem("savedContext", "");

alert('Failed to Resume with Error ' + responseJson.errCode);

alert(' Please restart application.');

}

});

## Logoff

The logoff routine is used to log off the user from that device.

###### Request Code Snippet

ReactRdna.logOff(value,(response) => {

if (response) {

console.log('immediate response is'+response[0].error);

}else{

console.log('immediate response is'+response[0].error);

}

});

###### Response Code Snippet

DeviceEventEmitter.addListener('onLogOff', function (e) {

console.log('immediate response is'+e.response);

var responseJson = JSON.parse(e.response);

if(responseJson.errCode == 0){

console.log('LogOff Successfull');

Obj.popToLoadView();

}else{

alert('Failed to Log-Off with Error '+responseJson.errCode);

}

});