**REL-ID API-SDK for iOS**

## **Get Started**

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).

This guide shows you how to create a new iOS project, include the REL-ID API-SDK and how to call it’s APIs .

Once you’ve installed the SDK, read on for the detailed API reference!

## **Prerequisites**

* Xcode 7.0 or higher
* Deployment target of 8.0 or higher

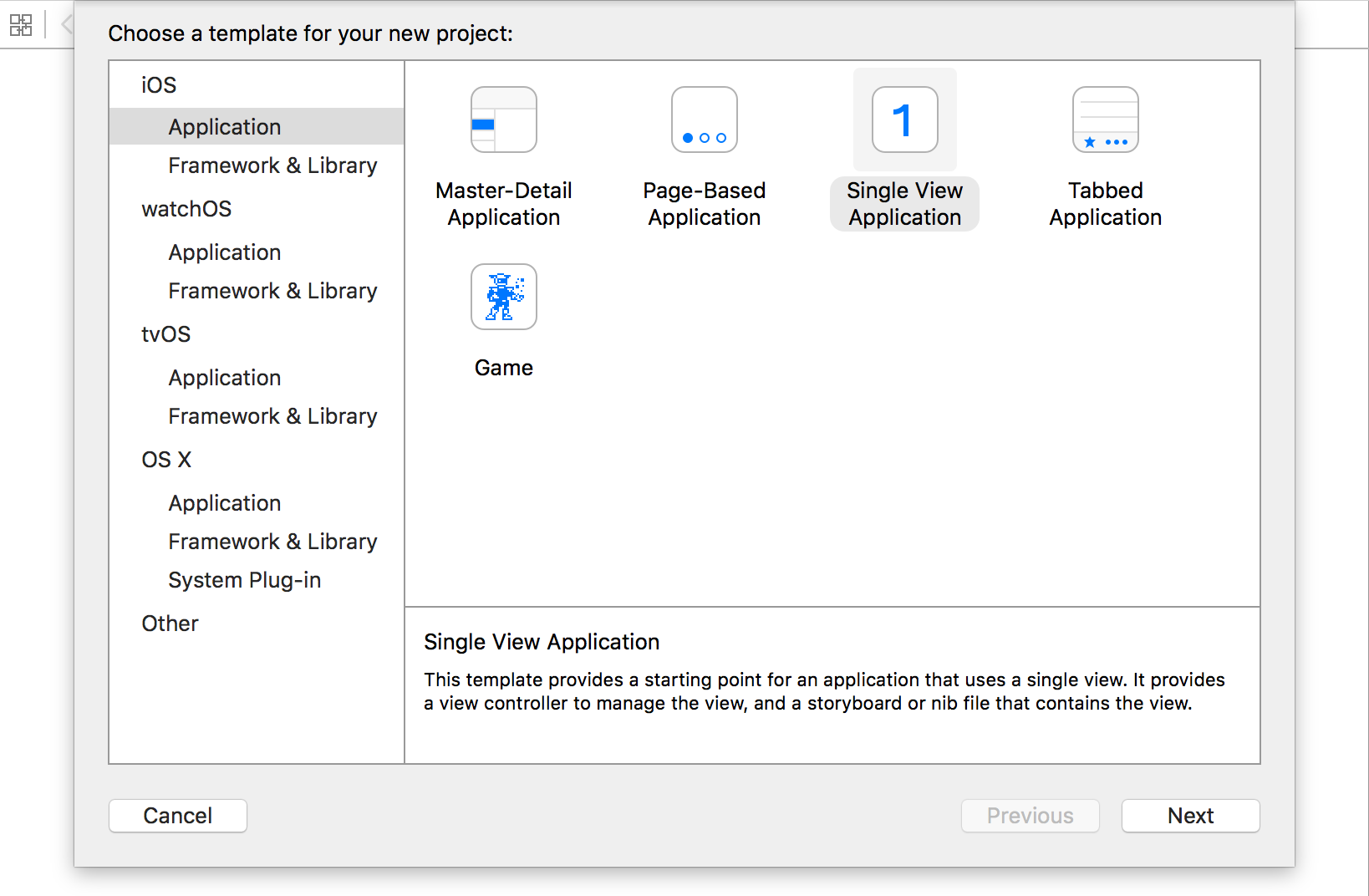
## **Creating a new project**

In this step, we'll create a new project in Xcode. If you don't already have Xcode running, go ahead and open it now.

#### 

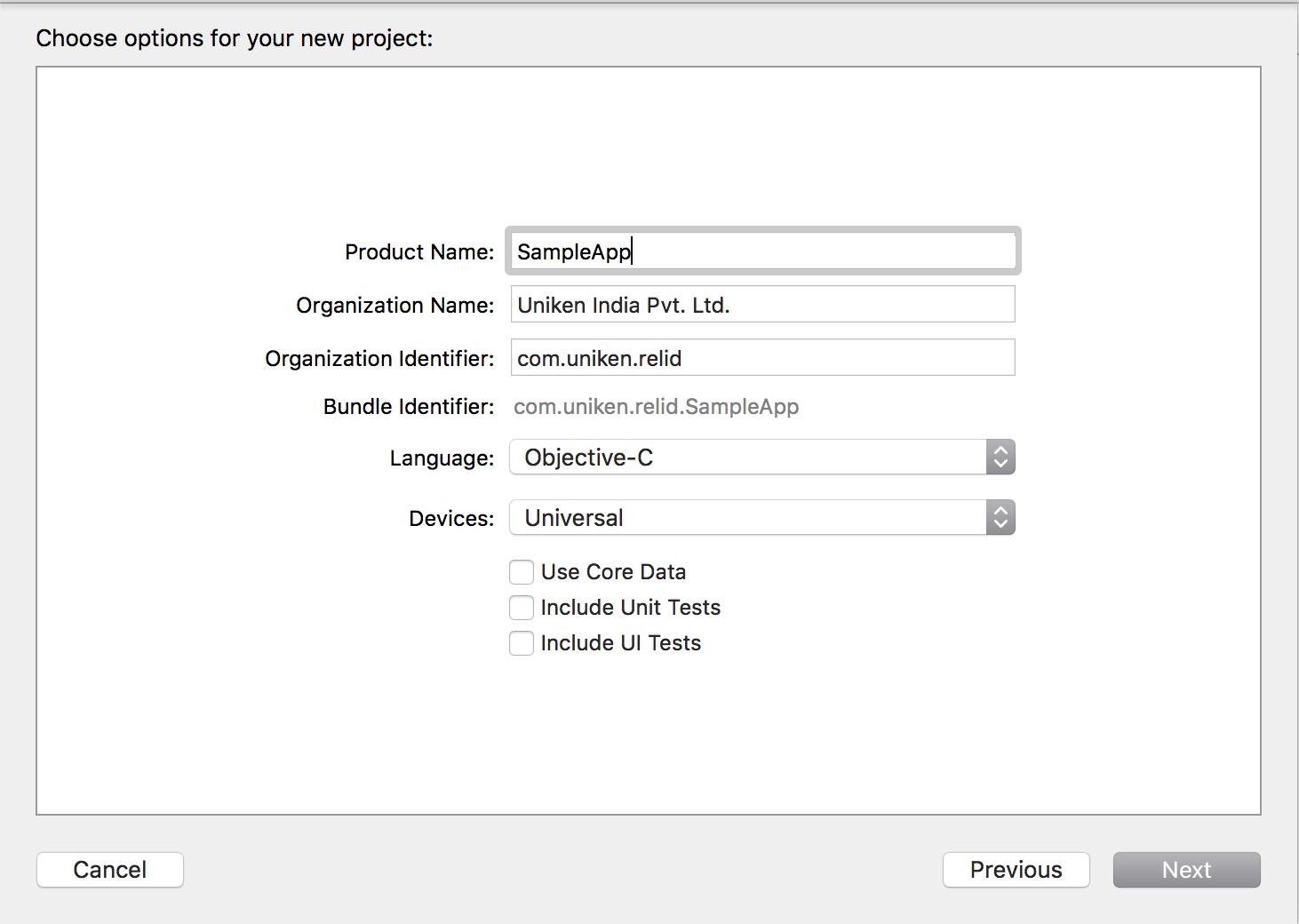
#### 

#### **Create a new Xcode project**



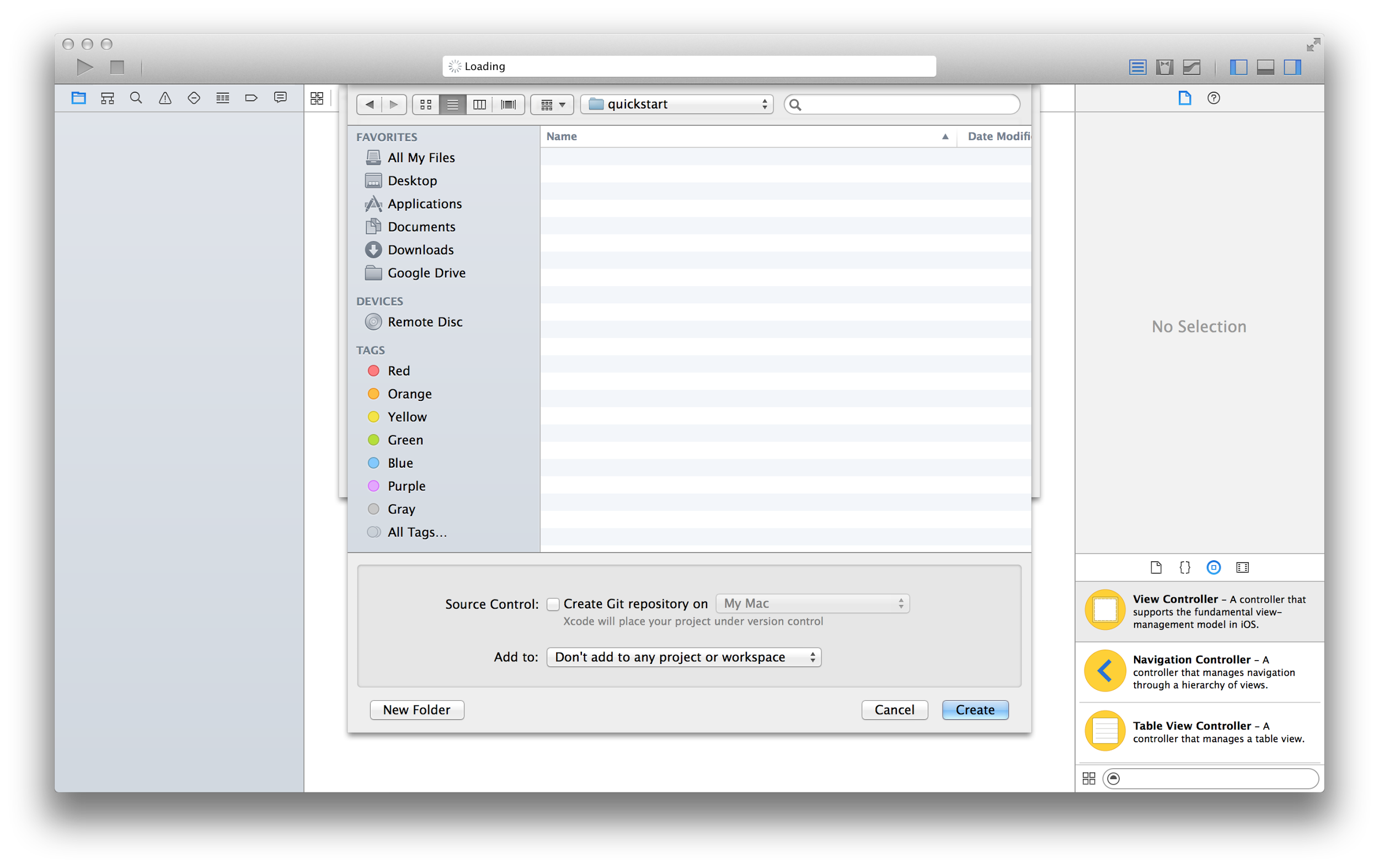
Navigate to **File > New > Project**. Select **Single View Application** under iOS apps, and click **Next**.

**Name your project**



Give your project the name "SampleApp". Choose "Objective-C" for the language. Then click **Next**.

**Select a project location**



Select a location for your project, and click **Create** to finish creating a new project.

#### 

#### **Build and run your new project**

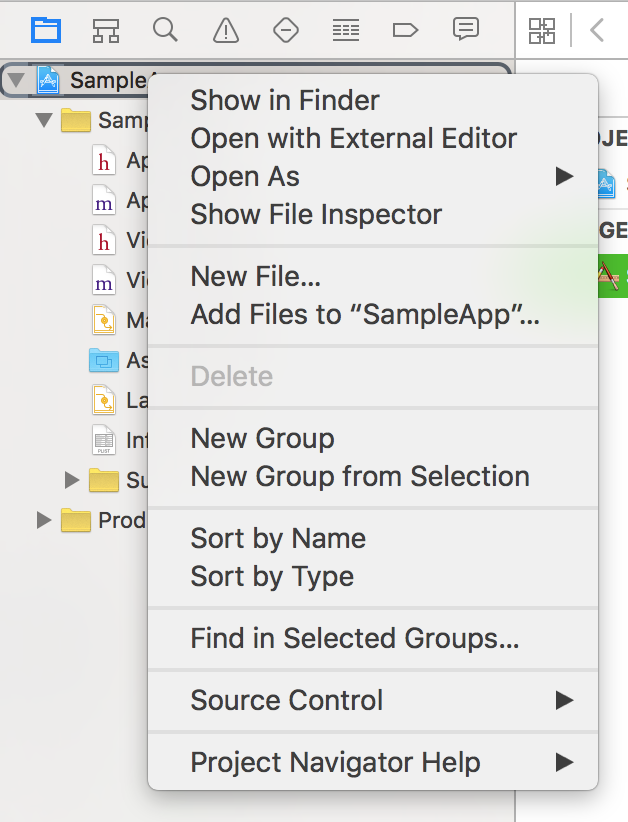
To use a simulator, navigate to **Product > Destination** and select an iPhone simulator. Then select **Product > Run** to verify your app builds and runs. The app shows just a blank white screen for now. But don't worry, we'll add content in the next steps.

**Adding the SDK to your Xcode project**

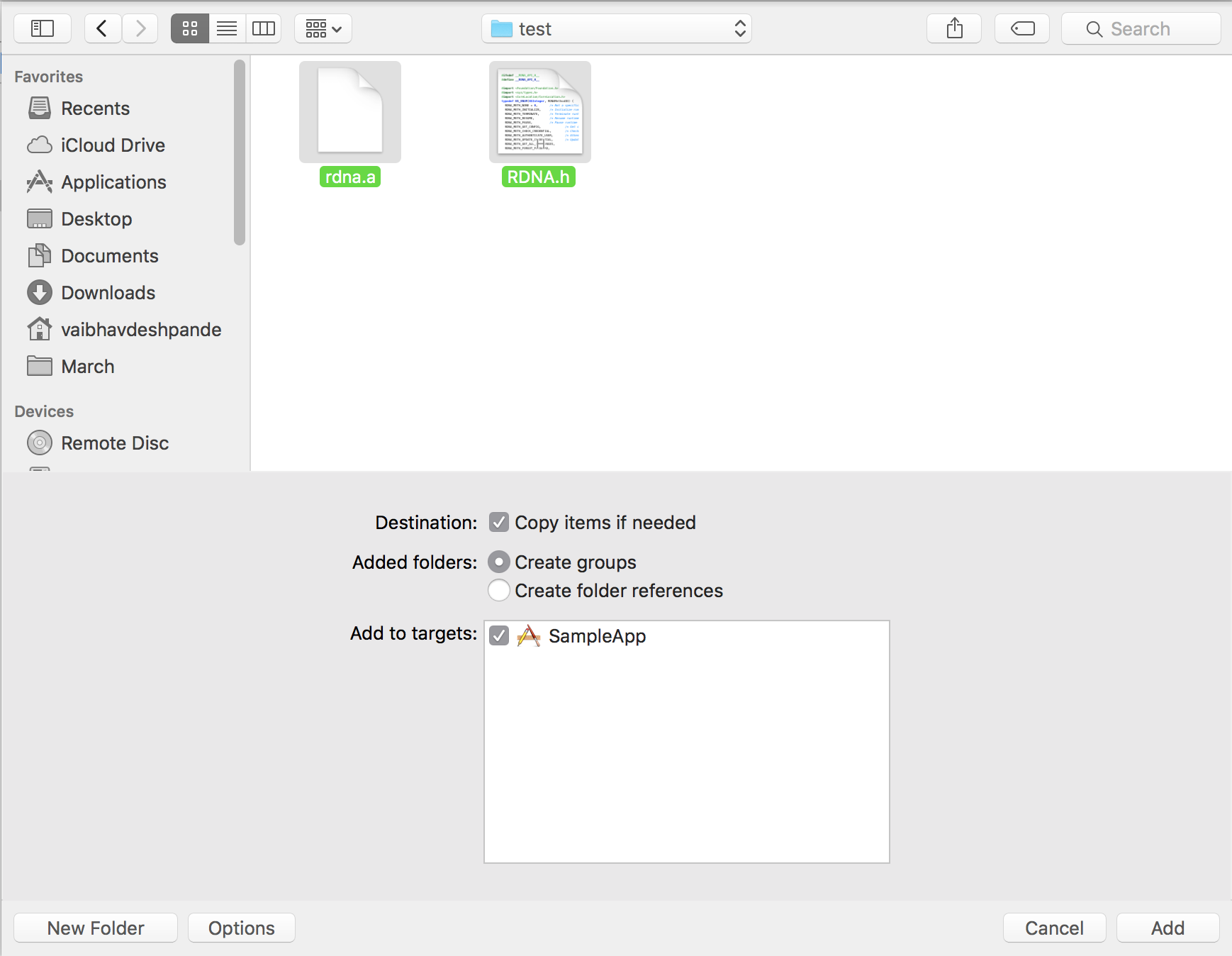
### **Manually, using the SDK download**

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 SampleApp project, and choose **Add Files To "SampleApp".**



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

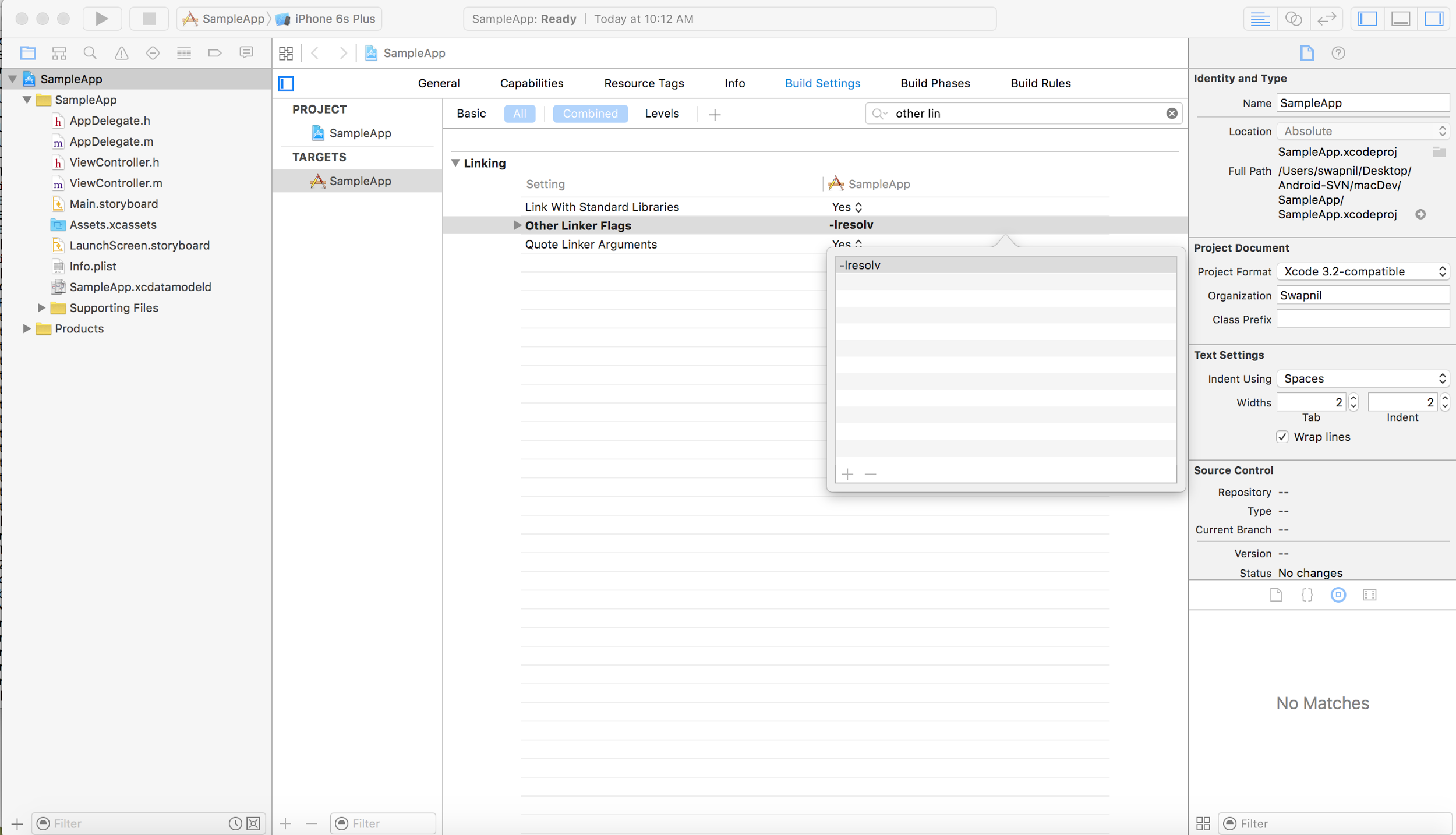
Once the SDK is referenced somewhere in your app, it automatically links these frameworks.

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.

## 

## **Sample code highlights**

**Initialize Routine**

This is the first routine that must be made to bootstrap the REL-ID API runtime up. The arguments to this routine are described in the below table. This routine starts up the API runtime (including a DNA instance), and in the process registers the API-client supplied callback routines with the API runtime context. This is a non-blocking routine, and when it returns, it will have initiated the process of creation of a REL-ID session in PRIMARY state - the progress of this operation is notified to the API-client application via the StatusUpdate callback routine supplied by it.

A reference to the context of the newly created API runtime is returned to the API-client. Please check the code snippet below to see how to call REL-ID API-SDK’s static (initialize:) method and implement its required callback methods, like (onInitializeCompleted).

//

// ViewController.m

// SampleApp

//

// Created by Uniken on 08/03/16.

// Copyright © 2016 Uniken India Pvt. Ltd. All rights reserved.

//

#import "ViewController.h"

#import "RDNA.h"

#import <CoreLocation/CoreLocation.h>

#define IS\_OS\_8\_OR\_LATER ([[[UIDevice currentDevice] systemVersion] floatValue] >= 8.0)

@interface ViewController () <RDNACallbacks, CLLocationManagerDelegate> {

CLLocationManager \*locationManagerObject;

}

@property (nonatomic, strong) RDNA \*rdnaObject;

@end

@interface Constant : NSObject

/\*\* RDNAClient parameters Constants \*\*\*/

extern NSString \*const kRdnaCipherSpecs;

extern NSString \*const kRdnaCipherSalt;

extern NSString \*const kRdnaHost;

extern uint16\_t const kRdnaPort;

extern NSString \*const kRdnaProxyHost;

@end

@implementation Constant

NSString \*const kRdnaCipherSpecs = @"AES/256/CFB/PKCS7Padding";

NSString \*const kRdnaCipherSalt = @"Rel-ID-Secure-IV";

NSString \*const kRdnaHost = @"127.0.0.1";

NSString \*const kRdnaProxyHost = @"127.0.0.1";

uint16\_t const kRdnaPort = 9080;

@end

@implementation ViewController

- (void)viewDidLoad {

[super viewDidLoad];

// Do any additional setup after loading the view, typically from a nib.

if (locationManagerObject == nil)

{

locationManagerObject = [[CLLocationManager alloc] init];

[locationManagerObject requestWhenInUseAuthorization];

if(IS\_OS\_8\_OR\_LATER) {

[locationManagerObject requestAlwaysAuthorization];

}

locationManagerObject.desiredAccuracy = kCLLocationAccuracyBest;

locationManagerObject.delegate = self;

}

[locationManagerObject startUpdatingLocation];

RDNA \*localRDNAObject;

int errorCode = [RDNA initialize:&localRDNAObject AgentInfo:@"test" Callbacks:(id<RDNACallbacks>)self GatewayHost:kRdnaHost GatewayPort:kRdnaPort CipherSpec:kRdnaCipherSpecs CipherSalt:kRdnaCipherSalt ProxySettings:nil AppContext:self];

if (errorCode > 0) {

NSLog(@"\n\n $$$$$ RDNA initialize invoked with errorcode : %d $$$$ \n\n", errorCode);

}

self.rdnaObject = localRDNAObject;

}

- (int)onInitializeCompleted:(RDNAStatusInit \*)status {

NSLog(@"\n\n $$$$$ Notify runtime status of client called reason {%ld : %d} $$$$ \n\n",(long)[RDNA getErrorInfo:status.errCode], status.errCode);

return status.errCode;

}

- (CLLocationManager \*)getLocationManager {

return locationManagerObject;

}

- (NSString \*)getApplicationVersion {

NSString \*version = [RDNA getSDKVersion];

return version;

}

- (NSString \*)getApplicationName {

NSDictionary \*infoDictionary = [[NSBundle mainBundle] infoDictionary];

NSString \*AppName = infoDictionary[(NSString \*) kCFBundleNameKey];

return AppName;

}

- (void)didReceiveMemoryWarning {

[super didReceiveMemoryWarning];

// Dispose of any resources that can be recreated.

}

@end

## **API methods and their description**

**Initialize method**

This interaction is governed by a single API routine (Initialize) invocation that sets the stage for all subsequent interactions.

Most importantly, this is the phase when the API runtime establishes an agent-authenticated session with the REL-ID platform backend and bootstraps the DNA for subsequent connectivity with both REL-ID platform services as well as the configured backend enterprise services

The following information is supplied by the API-client application to the initialization routine:

1. Agent information (available as a base64-encoded blob, upon provisioning a new agent REL-ID on a commercially licensed REL-ID Gateway Manager)
2. Callback methods/functions that the API runtime will use to communicate with the API-client application (status/error notifications, device context/fingerprint retrieval)
3. Network coordinates of the REL-ID Authentication Gateway (hostname/IP address and port number)
4. Privacy (encryption) specifications for the Data Privacy APIs - includes cipher specs and salt to use
5. Opaque reference to the API-client application context (never interpreted/modified by the API runtime, placeholder for application)
6. If applicable, proxy information for connecting through to the REL-ID Auth Gateway

*+ (int)initialize:(RDNA \*\_\_autoreleasing \*)ppRuntimeCtx*

*AgentInfo:(NSString \*)agentInfo*

*Callbacks:(id<RDNACallbacks>)callbacks*

*GatewayHost:(NSString \*)authGatewayHNIP*

*GatewayPort:(uint16\_t)authGatewayPORT*

*CipherSpec:(NSString \*)cipherSpec*

*CipherSalt:(NSString \*)cipherSalt*

*ProxySettings:(RDNAProxySettings \*)proxySettings*

*AppContext:(id)appCtx;*

**Description**

This method initializes the sdk by connecting out to given REL-ID server coordinate and creates REL-ID session.

**Parameters**

*agentInfo*

- Software identity information for the API-runtime to authenticate and establish primary session connectivity with the REL-ID platform backend.

*callbacks*

- Pointer to the callback object

*cipherSpec*

- Cipher mode to be used

*cipherSalt*

- Cipher salt

*proxySettings*

- Parent proxy settings of the application

*appCtx*

- Application context

*gwHNIP*

- Hostname/IP of the gateway server

*gwPort*

- Port of the gateway server

**Pause method**

On iOS, due to limited resource availability, the OS very often puts your application to sleep. When doing this, the OS gives the application a chance to save its state so that it may resume later when it is brought back to the foreground.

The pause API routine requires to be called in such an eventuality, so that the API runtime gets a opportunity to save its state and pass that state back to the API client application for saving to persistent storage on the device.

Typically the pause API routine must be last REL-ID API routine to be called, before saving the application state - along with the API runtime state.

*- (int)pauseRuntime:(NSMutableData \*\_\_autoreleasing \*)state;*

**Description**

pauseRuntime: method signals the RDNA to pause its execution and save the its state and return it to caller.

**Parameters**

*state*

- Output paramater

**Returns**

- Returns appropriate error code that occurred or success.

**Resume method**

When a mobile application has been paused, it must invoke the pause API routine, and save the returned API runtime state information, along with its own state.

When the same application is resumed, in its own resume sequence, it must invoke the resume API routine, passing in the runtime state that it had saved earlier, so that the API can reinitialize and resume its runtime operations.

Typically the resume API routine must be first REL-ID API routine to be called, immediately after loading the previously saved application state - along with the API runtime state.

*- (int)onResumeRuntime:(RDNAStatusResumeRuntime \*)status;*

**Description**

This method is used to signal the DNA thread to resume execution.

**Parameters**

*state*

- Output paramater

*callbacks*

- Pointer to the callback object

*proxySettings*

- Parent proxy settings of the application

*appCtx*

- Application context

**Returns**

- Returns appropriate error code that occurred or success.

**Terminate method**

The terminate API routine should be called during application shutdown in order to cleanly terminate the API runtime.

*- (int)terminate;*

**Description**

This method terminates the RDNA context.

**GetServiceByServiceName Method**

Retrieve the Service class object by looking up the unique logical name of the backend service (as configured in the REL-ID Gateway Manager)

*- (int)getServiceByServiceName:(NSString \*)serviceName*

*ServiceInfo:(RDNAService \*\_\_autoreleasing \*)service;*

**Description**

This method is used to get the details of the tunnelling service w.r.t. its name.

Note: - Its the apps resp. to cleanup the return service obj.

**Parameters**

*serviceName*

- Name of the service.

*service*

- The actual service object containg the service details will be filled in here.

**Returns**

- Appropriate error code that occurred or success.

**GetServiceByTargetCoordinate Method**

Retrieve one or more Service class object(s) by looking up the target (masked) coordinate, corresponding to the backend enterprise service. More than one services could be returned, depending on the access configuration for the context.

*- (int)getServiceByTargetCoordinate:(NSString \*)targetHNIP*

*TargetPort:(uint16\_t)targetPORT*

*ServicesInfo:(NSArray \*\_\_autoreleasing \*)services;*

**Description**

This method is used to get the details of the tunnelling service/services w.r.t. its destination co-ordinates.

Note: - Its the apps responsibility to cleanup the return service obj.

**Parameters**

*services*

- List of objects containing the service details will be filled in here.

*HNIP*

- Destination host-name/IP of the service

port

- Destination port of the service.

**Returns**

- Appropriate error code that occurred or success.

**GetAllServices Method**

Retrieve the Service class object(s) for all available services that are accessible via the current API-runtime context and the session within.

*- (int)getAllServices:(NSArray \*\_\_autoreleasing \*)services;*

**Description**

This method is used to get the details of all the services that are running.

**Parameters**

*services*

- List of objects containing the service details will be filled in here.

**Returns**

- Appropriate error code that occurred or success

**ServiceAccessStart Method**

Access to the service (i.e. the corresponding backend enterprise service) via the access port for the Service is started.

In case of TYPE\_PROXY port, the proxy facade of the DNA in the API-runtime listening on that port will start *tunneling* requests/data to the corresponding backend service.

In case of TYPE\_PORTF port, the corresponding forwarded TCP port is started in the DNA in the API-runtime, and made ready to accept connections from which data will be transparently forwarded to the corresponding backend service.

*- (int)serviceAccessStart:(const RDNAService \*)service;*

**Description**

This method is called to start a particular tunnelling service.

**Parameters**

*service*

- Object containg the service details. Name is mandatory and the only parameter to uniquely identify the service.

**Returns**

- Appropriate error code of the error that occurred or success.

**ServiceAccessStop Method**

Access to the service (i.e. the corresponding backend enterprise service) via the access port for the Service is stopped

In case of TYPE\_PROXY port, the proxy facade of the DNA in the API-runtime listening on that port will stop *tunneling* requests/data to the corresponding backend service and it will revert with an appropriate HTTP Proxy error code for further access to this service

In case of TYPE\_PORTF port, the corresponding forwarded TCP port is shutdown and closed in the DNA in the API-runtime, and connections to that port will no longer be accepted.

*- (int)serviceAccessStop:(const RDNAService \*)service;*

**Description**

This method is called to stop a particular tunnelling service.

**Parameters**

*service*

- Object containg the service details. Name is mandatory and the only parameter to uniquely identify the service.

**Returns**

- Appropriate error code of the error that occurred or success.

**ServiceAccessStartAll Method**

This method is used to start all the services that are configured on the connecting server.

*- (int)serviceAccessStartAll;*

**Description**

starts all the non-running tunnelling services

**Returns**

- Appropriate error code that occurred or success.

**ServiceAccessStopAll Method**

This method is used to stop all the services that are configured on the connecting server.

*- (int)serviceAccessStopAll;*

**Description**

stops all the running tunnelling services

Returns

- Appropriate error code that occurred or success.

**GetDefaultCipherSpec Method**

Get the default cipher spec used in the RDNA context

*- (int)getDefaultCipherSpec:(NSMutableString \*\_\_autoreleasing \*)cipherSpec;*

**Description**

This method is used to get the default cipher spec

**Parameters**

*cipherSpec*

[OUT] - default cipher spec set in the RDNA

**Returns**

- Appropriate error code of the error that occurred or success.

**GetDefaultCipherSalt Method**

Get the default cipher salt used in the RDNA context

*- (int)getDefaultCipherSalt:(NSMutableString \*\_\_autoreleasing \*)cipherSalt;*

**Description**

This method is used to get the default cipher spec salt

**Parameters**

*cipherSalt*

[OUT] - default cipher salt set in the RDNA

**Returns**

- Appropriate error code of the error that occurred or success.

**EncryptDataPacket Method**

Raw plaintext (unencrypted) data is supplied as a buffer of bytes.

This data is encrypted using keys as per specified privacy scope, and returned to calling API-client application.

*- (int)encryptDataPacket:(RDNAPrivacyScope)privacyScope*

*CipherSpec:(NSString \*)cipherSpec*

*CipherSalt:(NSString \*)cipherSalt*

*From:(NSData \*)plainText*

*Into:(NSMutableData \*\_\_autoreleasing \*)cipherText;*

**Description**

This method used to encrypt the data packet.

**Parameters**

*privacyScope*

- Privacy scope of the stream.

*cipherSpec*

- Cipher to be used.

*cipherSalt*

- Cipher salt.

*plainText*

- Input the plain string data packet.

*cipherText*

- Ouput the ecrypted data packet.

**Returns**

- Appropriate error code of the error that occurred.

**DecryptDataPacket Method**

Encrypted data is supplied as a buffer of bytes.

This data is decrypted using keys as per specified privacy scope, and returned to calling API-client application.

*- (int)decryptDataPacket:(RDNAPrivacyScope)privacyScope*

*CipherSpec:(NSString \*)cipherSpec*

*CipherSalt:(NSString \*)cipherSalt*

*From:(NSData \*)cipherText*

*Into:(NSMutableData \*\_\_autoreleasing \*)plainText;*

**Description**

This method decrypts the data packet provided and returns the plain string if successful else the appropriate error code.

**Parameters**

*privacyScope*

- Privacy scope of the stream.

*cipherSpec*

- Cipher to be used.

*cipherSalt*

- Cipher salt.

*dataPacketEncrypted*

- Input encrypted data packet.

*dataPacketDecrypted*

- Output decrypted data packet in plain string format.

**Returns**

- Appropriate error code of the error that occurred.

**EncryptHttpRequest Method**

HTTP request in plaintext (unencrypted) form is supplied as a buffer of bytes.

This request is encrypted using keys as per specified privacy scope, encoded appropriately, wrapped around in an HTTP request envelope and returned back to calling API-client application as another HTTP request.

*- (int)encryptHttpRequest:(RDNAPrivacyScope)privacyScope*

*CipherSpec:(NSString \*)cipherSpec*

*CipherSalt:(NSString \*)cipherSalt*

*From:(NSString \*)request*

*Into:*

*(NSMutableString \*\_\_autoreleasing \*)transformedRequest;*

**Description**

This method encrypts the original HTTP request and encapsulates it into a dummy httprequest with same destination as the original request.

**Parameters**

*cipherSpec*

- Cipher to be used.

*cipherSalt*

- Cipher salt.

*request*

- Input plaint text string containing the http request.

*transformedRequest*

- Ouput http request containing the original source and destination and conatains the payload which consists of the original http request in encrypted form.

requestLen

- Length of input plaintext HTTP request.

*scope*

- Privacy scope of the stream.

*transformedRequestLen*

- Length of encrypted output HTTP request.

**Returns**

- Appropriate error code of the error that occurred.

**DecryptHttpResponse Method**

HTTP response in encrypted form is supplied as a buffer of bytes.

This response is parsed, the embedded encrypted HTTP response is decoded, decrypted using keys as per specified scope, and returned back to calling API-client application as the original plaintext HTTP response.

*- (int)decryptHttpResponse:(RDNAPrivacyScope)privacyScope*

*CipherSpec:(NSString \*)cipherSpec*

*CipherSalt:(NSString \*)cipherSalt*

*From:(NSString \*)transformedResponse*

*Into:(NSMutableString \*\_\_autoreleasing \*)response;*

**Description**

This method decrypts the HTTP response and returns back the plain string original http response string.

**Parameters**

*privacyScope*

- Privacy scope of the stream.

cipherSpec

- Cipher to be used.

*cipherSalt*

- Cipher salt.

*transformedResponse*

- Input encrypted HTTP response string.

response

- Output decrypted to plain string HTTP response string.

**Returns**

- Appropriate error code of the error that occurred.

**GetSdkVersion Method**

Get the API-SDK version number used by the REL-ID API-SDK.

*+ (NSString \*)getSDKVersion;*

**Description**

This method is used to get the current SDK version used by the client.

**Returns**

returns the current SDK version used by the REL-ID API-SDK version in string format.

**GetErrorInfo Method**

Get the error information corresponding to an integer error code returned by any API. It returns back RDNAErrorID which gives brief information of the error occured.

*+ (RDNAErrorID)getErrorInfo:(int)errorCode;*

**Description**

This method is called to get enum value from error code.

**Returns**

appropriate enum.

**GetSessionID Method**

Get the session ID of the current initialized REL-ID session. Depending on the current state of the API-Runtime, the retrieved session ID will be either of the current application (PRIMARY) REL-ID session or of the current user (SECONDARY) REL-ID session.

*- (int)getSessionID:(NSMutableString \*\_\_autoreleasing \*)sessionID;*

**Description**

This method returns the session ID of the current initialized REL-ID session.

**Parameters**

*sessionID*

- Output parameter that is populated with session ID.

**Returns**

- Returns appropriate error code that occurred or success.

**GetAgentID Method**

using which the REL-ID session is initialized GetDeviceID Get the device ID of the current device using which the REL-ID session is initialized

*- (int)getAgentID:(NSMutableString \*\_\_autoreleasing \*)agentID;*

**Description**

This method returns the Agent ID using which the REL-ID session is initialized.

**Parameters**

*agentID*

- Output parameter that is populated with agent ID.

**Returns**

- Returns appropriate error code that occurred or success.

**CheckChallenge Method**

This routine submits one or more challenges like (secret questions and answer, verfication keys and access codes, passwords, responses to challenges etc) with the REL-ID backend, to authenticate the end-user. This can be used to - \* To check the user status - whether this user requires authentication on this device, what authentication to perform for this user, whether the user is blocked (on this device and/or app, or otherwise). \* To check the username and password. \* To check the additional authentication credentials such as secret answer, captcha, etc

The result is one of the following - \* FAILURE, <reason> \* CHALLENGE, <challenge> \* SUCCESS, <eurelid>

The result is informed to API-client via status update (core) / event notification (wrapper) callback routines.

Multiple successful invocations of this routine may be needed before receiving a SUCCESS, <eurelid> result from this routine, with the same challenge (retries) and/or different challenges (additional authentication).

*- (int)checkChallenges:(NSArray \*)challengeRequestArray*

*forUserID:(NSString \*)userID;*

**Description**

This method is used to send the response of challenges to server for authentication

**Parameters**

*userID*

- Unique user id.

*challenngeRequestArray*

- Challenges with response set to it.

**Returns**

- appropriate error code that occurred or success.

**UpdateChallenge Method**

The purpose of this routine is to update one or more credentials of the end-user. This routine can only be invoked if AuthenticateUser routine has been successfully completed. Pretty much all the relevant credentials of the end-user - secret question(s) and answer(s), one-time-use access code generation seeds, primary password(s), device binding to user-app etc - may be updated using this routine. The result is informed to API-client via status update (core) / event notification (wrapper) callback routines.

*- (int)updateChallenges:(NSArray \*)challengeRequestArray*

*forUserID:(NSString \*)userID;*

**Description**

updateChallenges - This method is used to update the challenges.

**Parameters**

*challengeRequestArray*

- List of challenges to update

*userID*

- Unique user id.

**Returns**

- appropriate error code that occurred or success.

**GetConfig Method**

The purpose of this routine is to get the configuration (if any) needed for API-Client from the REL-ID backend server so that the behavior of the API-Client can be made configurable in some use cases. For example - password policy to be validated when password is set/changed, customization settings, other application/user level settings, etc. The result is informed to API-client via status update (core) / event notification (wrapper) callback routines.

*- (int)getConfig:(NSString \*)userID;*

**Description**

This method can be used for any state primary or secondary of the RDNA context, and this is used to get the configuration setting from the server.

**Parameters**

*userID*

- Unique user id.

**Returns**

- Returns 0 on success or appropriate error code.

**ResetChallenge** **Method**

The purpose of this method is reset the authentication flow. If any error occurs in between when user is trying to authenticate itself with the server, reset challenge needs to be called to get to know to the server that the user is re-starting the authentication flow.

*- (int)resetChallenge;*

**Description**

This method is used to reset the challenge

**Returns**

- appropriate error code that occurred or success.

**GetAllChallenges Method**

The purpose of this method is to get all the authentication challenges for updating purpose.

*- (int)getAllChallenges:(NSString \*)userID;*

**Description**

getAllChallengesForUserID - This method is used to get all challenges

**Parameters**

userID

- UserID for needs to be provided for which user challenges need to be fetched.

**Returns**

- appropriate error code that occurred or success.

**LogOff Method**

The purpose of this method is to log off user session and switch back to application session.

*- (int)logOff:(NSString \*)userID;*

**Description**

logOff - This method is used to log off the user session.

**Parameters**

userID

- Unique user id.

**Returns**

- appropriate error code that occurred or success.

**ForgotPassword Method**

The purpose of this method is to inform server that user has forgotten the authentication credentials to get login and access to the services. Upon calling this API, server throws additional authentication challenges that are required.

*- (int)forgotPassword:(NSString \*)userID;*

**Description**

This method is responsible to handle use case when client is unable to reach for correct password.

**Parameters**

*userID*

- Unique user id.

**Returns**

- appropriate error code that occurred or success.

**SetDNSServers Method**

This method is used to set or add multiple DNS servers, for resolving IP address to domain names.

*- (int)setDNSServers:(NSArray \*)DNSServers;*

**Description**

This method is used to set the DNS servers manually.

**Parameters**

*DNSServers*

- This parameter is the array of the DNS server objects, which needs to be set as DNS servers.

**Returns**

- appropriate error code that occurred or success.

**Callback routines (types, structures, interfaces)**

This structure is supplied to the Initialize routing containing API-client application callback routines. These callback routines are invoked by the API runtime at different points in its execution - for updating status, for requesting the API-client application to supply information etc.

|  |  |  |
| --- | --- | --- |
| Callback Routine | Basic/Advanced | Description |
| onInitializeCompleted | Basic API | Invoked by the API runtime in order to update the API-client application of the progress of a previously invoked API routine, or state changes and exceptions encountered in general during the course of its execution.  *-(int)onInitializeCompleted:(RDNAStatusInit \*)status;*  **Description**  This is a asynchronous callback method that needs to be implemented by the end consumer, to get the initialization details after having invoked initialize to start API runtime.  **Parameters**  *status*  - This parameter contains the RDNAStatusInit class object.  **Returns**  - appropriate error code or success. |
| getApplicationFingerprint | Basic API | Invoked by the API runtime during initialization (session creation) in order to retrieve the application fingerprint, supplied by the API-client application to include in the device details.  The intent of this routine is to provide the application with an opportunity to identify itself so that the backend can check integrity of the application. To this end, it is recommended that the application provide strong checksums which can be matched/recorded at the backend.  *- (NSString \*)getApplicationFingerprint;*  **Description**  Retrieve the application fingerprint from consumer  **Returns**  - App's fingerprint data to be set into the device details |
| onTerminate | Basic API | *-(int)onTerminate:(RDNAStatusTerminate \*)status;*  **Description**  This is the callback method that needs to be implemented by the end consumer, to get terminate callback of terminate method.  **Parameters**  *status*  - This is the parameter that contains the RDNAStatusTerminate object.  **Returns**  - appropriate error code or success. |
| onPauseRuntime | Basic API | *-(int)onPauseRuntime:(RDNAStatusPauseRuntime \*)status;*  **Description**  This is the callback method that needs to be implemented by the end consumer, after having called pause the API runtime, after initialization is completed. The API runtime context if saved, to be able to call at resume.  **Parameters**  *status*  This is the parameter that contains the RDNAStatusPauseRuntime object.  **Returns**  Appropriate error code or success. |
| onResumeRuntime | Basic API | *-(int)onResumeRuntime:(RDNAStatusResumeRuntime \*)status;*  **Description**  This is the callback method that needs to implemented by the end consumer, after having callled resumeRuntime, which provides the status of resume.  **Parameters**  *status*  This is the parameter that contains the RDNAStatusResumeRuntime object.  **Returns**  Appropriate error code or success. |
| onConfigReceived | Advanced API | *-(int)onConfigRecieved:(RDNAStatusGetConfig \*)status;*  **Description**  This is the callback method that needs to be implemented by the end consumer, after having called getConfig API. In this callback all the configuration required by the client is provided.  **Parameters**  *status*  This is the parameter that contains the RDNAStatusGetConfig object.  **Returns**  Appropriate error code or success. |
| onChallengeReceived | Advanced API | *-(int)onChallengeRecieved:(RDNAStatusCheckChallenge \*)challengeStatus;*  **Description**  This is the callback method that needs to be implemented by the end consumer, after having called pause the checkChallenges method.  **Parameters**  *status*  This is the parameter that contains the RDNAStatusCheckChallenge object.  **Returns**  Appropriate error code or success. |
| onLogOff | Advanced API | *- (int)onLogOff:(RDNAStatusLogOff \*)status;*  **Description**  This is the callback method that needs to be implemented by the end consumer, after having called logOff of the API runtime.  **Parameters**  *status*  This is the parameter that contains the RDNAStatusLogOff object.  **Returns**  Appropriate error code or success. |
| getCredentials | Advanced API | *- (RDAIWACreds \*)getCredentials:(NSString \*)domainUrl;*  **Description**  This is a synchronous callback API is used to, get the credential information from end consumser for authenticating oneself on web server accessing through REL-ID API-SDK service.  **Parameters**  *status*  This is the parameter that contains the domainUrl string.  **Returns**  RDAIWACreds class object, that needs to be filled in by the end consumer to provided the credentials to the web server. |
| ShowLocationDialogue | Advanced API | *- (int)ShowLocationDailogue;*  **Description**  This is a synchornous callback method, that neeeds to be implemented by the end consumer, to when location service is disabled or off.  **Returns**  Appropriate error code or success. |
| getLocationManager | Advanced API | This is a synchronous callback method that needs to be implemented by the end consumer, to send the LocationManager to fetch location(i.e latitude, longitude and altitude).  *- (CLLocationManager \*)getLocationManager;*  **Description**  This is a synchronous callback method that needs to be implemented by the end consumer, to send the LocationManager to fetch location(i.e latitude, longitude and altitude).  **Returns**  locationManager instance. |