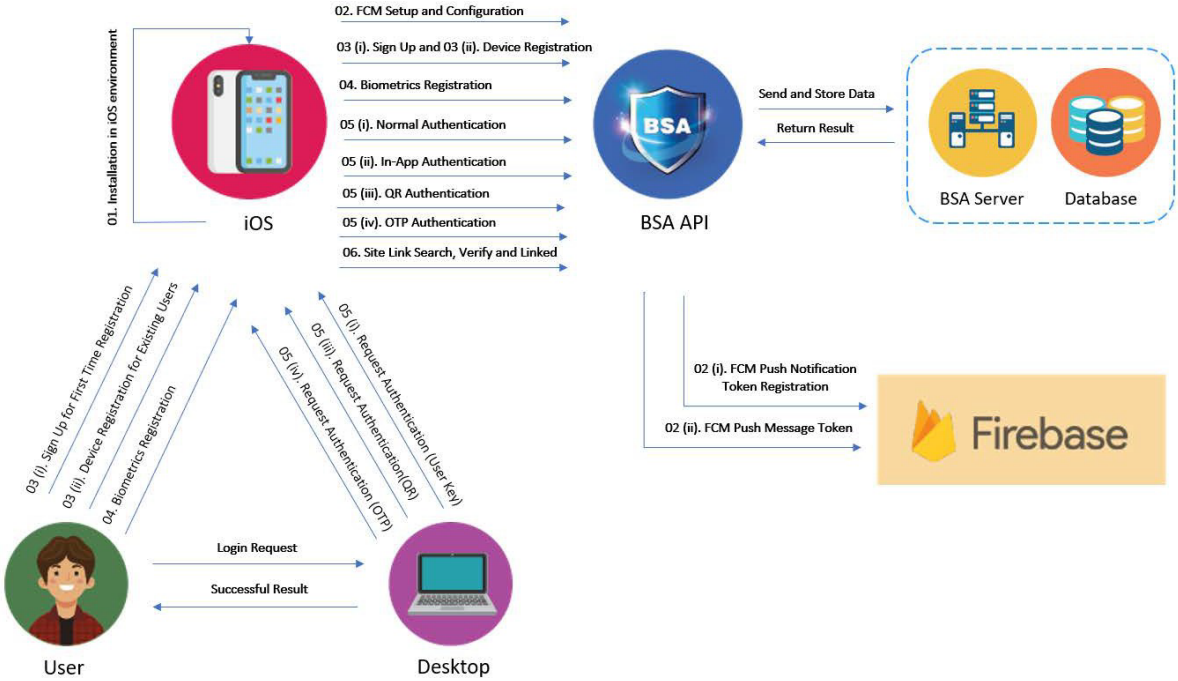
Getting Started

The BSA SDK, referred to as the 'iOS SDK' henceforth, equips developers with the necessary tools to integrate BSA authentication into their iOS applications. This document outlines the process of developing an iOS application while making the most effective use of the iOS SDK for seamless BSA authentication implementation.

Below is the flow that we are going to show:

1. Prerequisites
2. Installation
3. FCM Setup and Configuration
4. User Registration
5. Device Re-registration
6. Authentication
7. User Information Modification
8. Other APIs (Need for Access Token)
9. Other APIs (No need for Access Token)
10. Error Code



To fully operate the iOS SDK, requirement below are neccessary

- xCode version 14.0 and above
- iOS version 14.0 and above
- CocoaPods version 1.14.3 and above

- 0.12.1 (2024-01-12): Latest
  - Add TimeZone parameter to `BSA.APICaller.retrieveAuthHistory` function.
- 0.12.0 (2024-01-10)

    - Modify DeviceID management logic
    - Change in FCM token renewal conditions: Call renewal API once a day on first run
- 0.11.7 (2023-12-11)
    - Add token expiration event to API function that uses AccessToken
    - Added result callback to `BSA.APICaller.checkRequestedAuthExists` function.
- 0.11.6 (2023-11-10)
    - Added `onTokenExpired` callback to APIs that use authentication tokens.
    - Add `onCompleted` callback
    - Added simplified form of `verifyOTPNumber` API (using type)
- 0.11.3 (2023-10-27)
    - Delete the `authPlatform` parameter of `AutoAuthenticator`
    - Fix authentication bug in `AutoAuthenticator`
- 0.11.1 (2023-10-12)
    - Change `BSA.APICaller.requestOTPNumberSend` to `BSA.APICaller.requestSendOTPNumber`
    - Add `bundleIdentifier` parameter to `BSA.APICaller.requestSendOTPNumber`
- 0.11.0 (2023-10-05)
    - `AutoAuthenticator` bug fix
- 0.10.18 (2023-09-27)
    - First release version

Installation

i. Install with Cocoapods

iOS SDK can be installed with Cocoapods and Cocoapods version must be 1.13.4 and above. If Cocoapods is already installed, move to the directory of project that will use the iOS SDK, and proceed as following. Uses RxSwift's DisposeBag for management.

```
1. pod init - create the podfile
2. open podfile
3. add library below in the podfile

    target 'Foo' do
      .
      .

    # Add Alamofire and BSA_SDK. Be cautious as the version of BSA_SDK may change.
      pod 'Alamofire'
      pod 'BSA_SDK', '0.12.1', :source => 'https://github.com/fnsvalue-git/BSA-SDK-PodSpec.git'

      .
      .
    end

    # Add build settings
    post_install do |installer|
      installer.pods_project.targets.each do |target|
        target.build_configurations.each do |config|
          config.build_settings['IPHONEOS_DEPLOYMENT_TARGET'] = '14.0'
          config.build_settings['BUILD_LIBRARY_FOR_DISTRIBUTION'] = 'YES'
        end
      end
    end


4. come back to the terminal and do 'pod install'
```

Below libraries will be automatically installed with iOS SDK.

- CyrptoSwift
- SwiftyJSON
- StompClientLib
- DeviceKit
- SwiftOTP

1. Run pod repo update
2. Open Pod project's Build Settings, change BSA SDK -> iOS and Build Active Architecture Only -> No

3. Targets project > General > add files “Pods/BSA_SDK/BSA_SDK.xcframework” , under “Frameworks, Libraries, and Embedded Content” Section
    - The added `BSA_SDK.xcframework` must be set to `Do Not Embed`.
4. Call “BSA.testCall” to verify the SDK installation in the log

ii. Initialization

To use iOS SDK, there are things needed to be imported. Also, to initialize iOS SDK, add in the AppDelegate.swift file like below.

```
import BSA_SDK

@main
Class AppDelegate: UIResponder, UIApplicationDelegate {

func application (_application: UIApplication, didFinishLaunchingWithOptions
launchOptions: [UIApplication.LaunchOptionsKey: Any]?) -> Bool {
// Initialize the BSA_SDK
BSA_SDK.initialize(serverUrl: String, websocketUrl: String, securityKey: String,
removeToken:Bool = true )
}
}
```

Parameter

Key	Type	Description
serverUrl	String	Default URL for API calls
websocketUrl	String	Default WebSocket URL for node verification
securityKey	String	Key used for data generation within the SDK
removeToken	Bool	Decide whether to delete the AccessToken issued during authentication when initializing the SDK. Default is true

In order to fully utilize the iOS SDK, **the client application must be registered to generate the client key.**

iii. Device Verification

API: BSA.APICaller.checkDevice

```
BSA.APICaller.checkDevice(bundleIdentifier: "com.fnsv.sdk",
                          clientKey: "a1hrg2xv...",
onSuccessByRegistered: { [weak self] data in

    // Registered Devices
```

```
}, onSuccessByNotRegistered: {

    // Unregistered Devices

}, onFailed: { [weak self] rtCode, resultMessage in

    // Invocation Failure

}, onError: { [weak self] error in

    // Invocation Error

}, onTotalLog: { text in

    // Invocation Log

}, disposeBag: disposeBag)
```

Parameter

```
BSA.APICaller.checkDevice(
        bundleIdentifier: String,
        clientKey: String,
        onSuccessByRegistered: @escaping (BSA.JSONData.CheckDeviceData) -> (),
        onSuccessByNotRegistered: @escaping () -> (),
        onFailed: @escaping OnFailed,
        onError: @escaping OnError,
        onTotalLog: @escaping OnTotalLog,
        disposeBag: DisposeBag )
```

Key	Type	Description
bundleIdentifier	String	App's bundleIdentifier
clientKey	String	Client's unique key
onSuccessByRegistered	(BSA.JSONData.CheckDeviceData) -> ()	Called when the device is registered. Returns CheckDeviceData.
onSuccessByNotRegistered	() -> ()	Called when the device is not registered.
onFailed	OnFailed	Callback function called on API processing failure.
onError	OnError	Callback function called on error.
onTotalLog	OnTotalLog	Callback function to receive all logs generated during API calls.

Key	Type	Description
disposeBag	DisposeBag	DisposeBag variable to manage RxSwift Disposable objects.

- If the device is registered, you can find brief User information in **BSA.JSONData.CheckDeviceData**, so use that information.
- If the device is not registered, you should proceed with SignUp or Register New Device.
  The user's last Local Authentication Type can be determined after calling this function by using **BSA.loadLocalAuthType()**.

BSA.JSONData.CheckDeviceData

Key	Type	Description
userKey	String	UserID
name	String	User Name
phoneNum	String	Phone Number
email	String	Email Address
uptDt	String	Modification Date

FCM Setup and Configuration

There can be a circumstance when the user requests BSA authentication from a non-mobile platform such as the web. To receive the push notification in the mobile app that has implemented the iOS SDK, FCM registration and the token update feature is necessary.

i. FCM Server Key

In order to receive the push notifications, the **Server Key** generated by Firebase Cloud Messaging (FCM) must be sent together with the client issuance request. The FCM **Server Key** can be found in the process below.

1. Create or load the project in the Firebase Console.
2. In top left, click on " ⚙ " icon located beside Project Overview and click **Project Settings**
3. In the tabs, click on **Cloud Messaging** tab.
4. Copy **Server Key** from Cloud Messaging API. If you do not have the server key yet, You can enable Cloud Messaging API and generate the key. More about Cloud Messaging API can be find in the linked documentation.

It is essential to note that the FCM server key is a crucial component of the process. This key plays a vital role in ensuring the smooth registration of clients on the platform.

ii. FCM Push Notification Token Registration

Registering the FCM push token is pivotal to enable effective notification of authentication requests.

It should be called within **func messaging(_ messaging: Messaging**, **didReceiveRegistrationToken fcmToken: String?)** in **Firebase's MessagingDelegate**.

Registration and update should only proceed if the registered FCM token and the FCM token received as a parameter are different.

The FCM push token may undergo changes due to factors like expiration. To accommodate such scenarios, it's important to update the FCM push token by utilizing the registerPushToken() function. This ensures that the most current and valid token is in use for seamless notification functionality.

```
- AppDelegate.swift

extension AppDelegate : MessagingDelegate {

    func messaging(_ messaging: Messaging, didReceiveRegistrationToken fcmToken:
String?) {

        // Mandatory SDK Call: FCM Token Registration and Refresh
        BSA.FCM.tokenUpdate(fcmToken: fcmToken)


    }
```

```
}
```

Parameter

Key	Value	Description
fcmToken	String	FCM token for receiving authentication push notifications.

iii. Load FCM Data

It should be called from the **UNUserNotificationCenterDelegate** function (see code).

When calling **BSA.FCM.load**, if parsing of the data in the notification is successful, the callback registered with **BSA.FCM.registerPushNotificationProcessor(callBack:)** is immediately invoked.

```
- AppDelegate.swift

extension AppDelegate : UNUserNotificationCenterDelegate {

    //When the device OS version is higher than iOS 13, it will receive the
notification and handle by this method
    func userNotificationCenter(_ center: UNUserNotificationCenter,
                                willPresent notification: UNNotification,
                                withCompletionHandler completionHandler: @escaping
(UNNotificationPresentationOptions) -> Void) {

        let content = notification.request.content
        let userInfo = content.userInfo
        UIApplication.shared.applicationIconBadgeNumber = 0
        UNUserNotificationCenter.current().removeAllPendingNotificationRequests()
        Messaging.messaging().appDidReceiveMessage(userInfo)



        // Mandatory SDK Call: Loading Push Data
        if BSA.FCM.load(notification: notification) {

            print("userNotificationCenter willPresent : push load success")

            completionHandler([.sound])

            return

        }



        completionHandler([.sound, .badge, .banner])
```

```
    }


    //When the device OS version is higher than iOS 13 and user clicks the push
notification, it will be handled by this method
    func userNotificationCenter(_ center: UNUserNotificationCenter,
                                didReceive response: UNNotificationResponse,
                                withCompletionHandler completionHandler: @escaping
() -> Void) {


        // Mandatory SDK Call: Loading Push Data
        if BSA.FCM.load(notification: response.notification) {

            print("userNotificationCenter didReceive : push load success")

            completionHandler()

            return

        }




        defer {
            completionHandler()
        }

        guard response.actionIdentifier == UNNotificationDefaultActionIdentifier
else {
            return
        }


    }

}
```

Parameter

BSA.FCM.load(notification: UNNotification)

Key	Value	Description
notification	UNNotification	Notification with Push Data.

iv. Detect Authentication Requests

To handle authentication requests even when the app is entered by the user's own action or through screen transitions without launching the notification, **BSA.FCM.checkRequestedAuthExists()** must be called in **SceneDelegate**'s **sceneWillEnterForeground** when an authentication push is received.

```
- SceneDelegate.swift

func sceneWillEnterForeground(_ scene: UIScene) {

    // Mandatory SDK Call: Check for Authentication Requests When Switching from
Background to Foreground
    BSA.FCM.checkRequestedAuthExists { isExisted in

        // Send UI update notification when necessary
        let notification = Notification(name: NAME_CHECK_REQ_AUTH_EXISTS,
                                        object: nil,
                                        userInfo: ["isExisted": isExisted])

        NotificationCenter.default.post(notification)

    }

}
```

vi. Use FCM Push Data

If parsing is successful in **BSA.FCM.load(notification: UNNotification)**, this function calls the registered callback function.
BSA.FCM.PushData is used for authentication logic and should be stored separately or processed for authentication logic.

```
// Mandatory SDK Call: Register Callback for Push Message Loading Completion
BSA.FCM.registerPushNotificationProcessor { data in

    // Check Push Type
    switch data.target {
    case .Request:
        // Display Alert for Push Authentication Request
        // showPushAuthenticationAlert(.....)
        // Request Authentication Logic Using Data

    case .Change:
        // Call Change When OTP Input Is Successful During OTP Authentication
Process


    default:

        PCGPrint.d("push process : skip")
```

```
    }
}
```

Parameter

```
public typealias Callback = (BSA.FCM.PushData)->()

BSA.FCM.registerPushNotificationProcessor(_ callback: @escaping Callback)
```

BSA.FCM.registerPushNotificationProcessor

Key	Value	Description
callback	Callback	Callback function that receives Push Data as a parameter.

BSA.SDK.PushData

Mainly used values in BSA.SDK.PushData

Key	Value	Description
target	PushType	Types of push notifications: Request, Success, Cancel, Failed, Change.
clientName	String	Name of the client requesting authentication.
client_key	String	ClientKey of the client requesting authentication.
siteUrl	String	Client's website address.
block_key	String	Block key to be used for authentication.
channel_key	String	Channel key to be used for authentication.
otpAuth	Bool	Whether it is OTP authentication or not.
identifier	String	An identifier that can be used for control in NotificationCenter.

User Registration

iOS SDK provides featured to register and use BSA_SDK by using basic information.

i. Duplicated Registered Information (Check email and phone number)

Check if there is any duplicated email and phone number before BSA registration process. It requests API by **BSA.APICaller.checkDuplicateUserData** of BSA_SDK. Whether user information is unique or not can be identified upon the **verifyType**.

To check for duplicate emails, set verifyType to Email and put the email address in verifyData.

To check for duplicate phone numbers, set verifyType to PhoneNumber and put the phone number in verifyData.

Example

```
BSA.APICaller.checkDuplicateUserData(verifyData: "user@email.com",
                                     verifyType: .Email,
onNotDulplicateUser: { [weak self] in

    // When Not Duplicate


}, onDulplicateUser: { [weak self] in

    // When Duplicate



}, onFailed: { [weak self] rtCode, resultMessage in

    // Invocation Failure

}, onError: { [weak self] error in

    // Invocation Error

}, onTotalLog: { text in

    // Invocation Log

}, onCompleted: {

    // invocation completed

}, disposeBag: disposeBag)
```

Parameter

```
BSA.APICaller.checkDuplicateUserData(verifyData: String,
                                     verifyType: BSA.Enums.VerifyType,
                                     onNotDulplicateUser: @escaping () -> (),
                                     onDulplicateUser: @escaping () -> (),
                                     onFailed: @escaping OnFailed,
                                     onError: @escaping OnError,
                                     onTotalLog: @escaping OnTotalLog? = nil,
                                     onCompleted: @escaping onCompleted? = nil,
                                     disposeBag: DisposeBag )
```

Parameter	Type	Description
verifyData	String	Data for duplicate checking (email, phoneNumber).
verifyType	BSA.Enums.VerifyType	Duplicate check type (Email, PhoneNumber).
onNotDuplicateUser	() -> ()	Callback function called when data is not duplicate.
onDuplicateUser	() -> ()	Callback function called when data is duplicate.
onFailed	OnFailed	Callback function called on API processing failure.
onError	OnError	Callback function called on error.
onTotalLog	OnTotalLog	Callback function to receive all logs generated during API calls.
onCompleted	onCompleted	Called at the end after all processes are completed.
disposeBag	DisposeBag	DisposeBag variable to manage RxSwift Disposable objects.

ii. OTP Number Request

Use the clientKey obtained when requesting SDK usage. If the call is successful, you can receive the OTP number via email/sms.

API : BSA.APICaller.requestOTPNumberSend
Example

```
BSA.APICaller.requestSendOTPNumber(sendType: .Email,
                                   sendTarget: "user@email.com",
                                   clientKey: "a1hrg2xv...",
                                   bundleIdentifier: "com.your.app",
onSuccess: {

    // Successful Request for OTP Number Transmission

}, onFailed: { [weak self] rtCode, resultMessage in
```

```
    // Invocation Failure
}, onError: { [weak self] error in
    // Invocation Error
}, onTotalLog: { text in
    // Invocation Log
}, onCompleted: {
    // invocation completed
}, disposeBag: disposeBag)
```

Parameter

```
BSA.APICaller.requestSendOTPNumber(sendType: BSA.APICaller.OTPSendType,
                                   sendTarget: String,
                                   clientKey: String,
                                   bundleIdentifier: String,
                                   onSuccess: @escaping () -> (),
                                   onFailed: @escaping OnFailed,
                                   onError: @escaping OnError,
                                   onTotalLog: @escaping OnTotalLog? = nil,
                                   onCompleted: @escaping onCompleted? = nil,
                                   disposeBag: DisposeBag )
```

Parameter	Type	Description
sendType	BSA.APICaller.OTPSendType	Platform to receive OTP number (Email, SMS)
sendTarget	String	Address for OTP number type
bundleIdentifier	String	App's bundleIdentifier
onSuccess	() -> ()	Callback called on success.
onFailed	OnFailed	Callback function called on API processing failure.
onError	OnError	Callback function called on error.
onTotalLog	OnTotalLog	Callback function to receive all logs generated during API calls.
onCompleted	onCompleted	Called at the end after all processes are completed.
disposeBag	DisposeBag	DisposeBag variable to manage RxSwift Disposable objects.

iii. OTP Number Validation

Enter the OTP number received by email/sms, call the API, and if successful, receive the **disposeToken** string. **disposeToken** is used to check whether the user is the correct user during SignUp.

API : BSA.APICaller.verifyOTPNumber
Example

```
BSA.APICaller.verifyOTPNumber(sendType: BSA.APICaller.OTPSendType,
                              sendTarget: String,
                              otpNumber: "298336",
                              clientKey: "a1hrg2xv...",
onValidOTPNumber: { [weak self] disposeToken in

    // OTP Number Matches

    // Token Storage
    self?.data.disposeToken = disposeToken

}, onInvalidOTPNumber: {

    // OTP Number Doesn't Match

}, onFailed: { [weak self] rtCode, resultMessage in

    // Invocation Failure

}, onError: { [weak self] error in

    // Invocation Error

}, onTotalLog: { text in

    // Invocation Log

}, onCompleted: {

    // invocation completed

}, disposeBag: disposeBag)
```

Parameter

```
BSA.APICaller.verifyOTPNumber(sendType: BSA.APICaller.OTPSendType,
                              sendTarget: String,
                               otpNumber: String,
                               clientKey: String,
                               onValidOTPNumber: @escaping (String) -> (),
                               onInvalidOTPNumber: @escaping () -> (),
                               onFailed: @escaping OnFailed,
                               onError: @escaping OnError,
                               onTotalLog: @escaping OnTotalLog? = nil,
```

```
                                onCompleted: @escaping onCompleted? = nil,
                                disposeBag: DisposeBag )
```

Parameter	Type	Description
sendType	BSA.APICaller.OTPSendType	Platform to receive OTP number (Email, SMS)
sendTarget	String	Address for OTP number type
otpNumber	String	Received OTP number.
clientKey	String	Client's unique Key
onValidOTPNumber	(String) -> ()	Callback called when OTP numbers match. The disposeToken value passed as a parameter is used in SignUp.
onInvalidOTPNumber	() -> ()	Callback called when OTP numbers don't match.
onFailed	OnFailed	Callback function called on API processing failure.
onError	OnError	Callback function called on error.
onTotalLog	OnTotalLog	Callback function to receive all logs generated during API calls.
onCompleted	onCompleted	Called at the end after all processes are completed.
disposeBag	DisposeBag	DisposeBag variable to manage RxSwift Disposable objects.

iv. Check Available ID

Used to validate the UserID required during SignUp.

API : BSA.APICaller.checkUserKeyAvailable
Example

```
BSA.APICaller.checkUserKeyAvailable(userKey: "user1",
onUserKeyAvailable: {

    // User Key Is Usable

}, onUserKeyNotAvailable: {
    // User Key Is Not Usable

}, onFailed: { [weak self] rtCode, resultMessage in

    // Invocation Failure

}, onError: { [weak self] error in

    // Invocation Error
```

```
}, onTotalLog: { text in

    // Invocation Log

}, onCompleted: {

    // invocation completed

}, disposeBag: disposeBag)
```

Parameter

```
BSA.APICaller.checkUserKeyAvailable(userKey: String,
                                    onUserKeyAvailable: @escaping () -> (),
                                    onUserKeyNotAvailable: @escaping () -> (),
                                    onFailed: @escaping OnFailed,
                                    onError: @escaping OnError,
                                    onTotalLog: @escaping OnTotalLog? = nil,
                                    onCompleted: @escaping onCompleted? = nil,
                                    disposeBag: DisposeBag )
```

Parameter	Type	Description
userKey	String	UserKey to check availability.
onUserKeyAvailable	() -> ()	Callback called if the UserKey is available.
onUserKeyNotAvailable	() -> ()	Callback called if the UserKey is unavailable.
onFailed	OnFailed	Callback function called on API processing failure.
onError	OnError	Callback function called on error.
onTotalLog	OnTotalLog	Callback function to receive all logs generated during API calls.
onCompleted	onCompleted	Called at the end after all processes are completed.
disposeBag	DisposeBag	DisposeBag variable to manage RxSwift Disposable objects.

vi. Local Authentication

Used to validate the UserID required during SignUp.

When called, it performs biometric and passcode authentication.
For SignUp, there is no saved **DomainState** for biometric authentication, so **checkDomainStateChanged** should be set to false.
Local Authentication must succeed to invoke **BSA.APICAller.requestSignUp**.
The callback called with onResult runs on a background thread, so UI operations should be done on the main thread.

API : BSA.LocalAuth.startSimpleAuth

Example

```
BSA.LocalAuth.startSimpleAuth(localAuthType: .BIOMETRIC,
                              reason: "Authentication is required to verify your
identity",
                              checkDomainStateChanged: false,
                              usePasscodeWhenFailed: false,
onSuccess: {

    // onSuccess operates on a background thread, so UI-related tasks should be
performed on the main thread.
    DispatchQueue.main.async {

        // Local Authentication Successful

    }

}, onError: { errorCode in

    // onError operates on a background thread, so UI-related tasks should be
performed on the main thread.
    DispatchQueue.main.async {

        // Local Authentication Failed

        switch errorCode {
        case .DOMAIN_STATE_CHANGED:
            // Biometric authentication info has been changed
        case .BIOMETRIC_NOT_AVILABLE
        case .BIOMETRIC_NOT_SUPPORT_HARDWARE
        case .BIOMETRIC_LOCK_OUT:
            // Biometric authentication not available
        case .BIOMETRIC_NOT_ENROLLED_DEVICE:
            // Biometric authentication info is not on device
        case .PASSCODE_NOT_ENROLLED_DEVICE:
            //  Passcode  not  on  device
        case .USER_CANCEL, .USER_FALLBACK:
            // User cancels authentication
        case .AUTH_FAILED:
            // Authentication failed
        default:
            // Other errors
        }

    }

})
```

Parameter

```
BSA.LocalAuth.startSimpleAuth(
                        localAuthType: BSA.Enums.LocalAuthType,
                        reason: String,
                        checkDomainStateChanged: Bool,
                        usePasscodeWhenFailed: Bool,
                        onSuccess: @escaping ()->(),
                        onError: @escaping (BSA.LocalAuthHelper.ErrorCode?)->())
```

Parameter	Type	Description
localAuthType	BSA.Enums.LocalAuthType	Local authentication type
reason	String	Message displayed on OS during biometric authentication
checkDomainStateChanged	Bool	Determine whether to detect biometric changes
usePasscodeWhenFailed	Bool	Decide whether to use a passcode when biometric authentication fails
onSuccess	()->()	Local authentication processing result callback
onError	(BSA.LocalAuthHelper.ErrorCode?)->()	Local authentication processing result callback

vii. Sign Up

userData must be filled with complete data when making the call.

API : BSA.APICaller.requestSignUp
Example

```
BSA.APICaller.requestSignUp(userData: data.userData!,
onSuccess: {
    // Success SignUp

}, onFailed: { [weak self] rtCode, resultMessage in

    // Invocation Failure

}, onError: { [weak self] error in

    // Invocation Error

}, onTotalLog: { text in

    // Invocation Log

}, onCompleted: {
```

```
    // invocation completed

}, disposeBag: disposeBag)
```

Parameter

```
BSA.APICaller.requestSignUp(userData: BSA.SignUpData.UserData,
                                    onSuccess: @escaping () -> (),
                                    onFailed: @escaping OnFailed,
                                    onError: @escaping OnError,
                                    onTotalLog: @escaping OnTotalLog? = nil,
                                    onCompleted: @escaping onCompleted? = nil,
                                    disposeBag: DisposeBag )
```

Parameter	Type	Description
userData	BSA.SignUpData.UserData	Data required for SignUp. Fill all fields and make the call.
onSuccess	() -> ()	Callback called on SignUp success.
onFailed	OnFailed	Callback function called on API processing failure.
onError	OnError	Callback function called on error.
onTotalLog	OnTotalLog	Callback function to receive all logs generated during API calls.
onCompleted	onCompleted	Called at the end after all processes are completed.
disposeBag	DisposeBag	DisposeBag variable to manage RxSwift Disposable objects.

BSA.SignUpData.UserData

Name	Type	Description
appPackage	String	App's bundleIdentifier
clientKey	String	Client's unique Key
appVersion	String	App version (e.g., 1.1.1)
token	String	FCM token value.
email	String	Email address.
phoneNumber	String	Phone number.
userKey	String	UserID
name	String	User name
authType	BSA.Enums.LocalAuthType	Local Authentication Type (NONE, BIOMETRIC, PASSCODE)

Name	Type	Description
os	BSA.Enums.OSType	os type (fixed to iOS)
disposeToken	String	Token received upon OTP authentication success.
agreeGccs	Bool	Agreement to BSA Terms of Use.
agreePerson	Bool	Agreement to Privacy Policy.
agreeDevice	Bool	Agreement to Device Information Policy.

If SignUp is successful, you can retrieve information using the following functions.

- BSA.loadTOTPSecretKey
- BSA.loadDeviceID
- BSA.loadFCMToken
- BSA.loadUserKey
- BSA.loadLocalAuthType

Device Re-registration

iOS SDK provides featured to re-register device and use BSA SDK by using basic information.

i. User Information Verification & OTP Number Request

userKey, **name**, **verifyDatamust** all be valid for OTP to be sent. In all other cases, it will report an error through **onFailed**.
The L**ocalAuthType** received on success represents the **Local Authentication Type** last used by the user and must be authenticated before making a Register New Device request.

Example

```
BSA.APICaller.requestSendOTPNumberForRegisterDevice(
                                        clientKey: "a1hrg2xv...",
                                        userKey: "user1",
                                        name: "username1",
                                        sendType: .Email,
                                        sendTarget: "user@email.com",
                                        bundleIdentifier: "com.your.app",
onSuccess: { [weak self] localAuthType in
    // Successful Invocation

    // Local Authentication Type of Registered User
    self?.data.existingUserData?.authType = localAuthType

}, onFailed: { [weak self] rtCode, resultMessage in

    // Invocation Failure

}, onError: { [weak self] error in

    // Invocation Error

}, onTotalLog: { text in

    // Invocation Log

}, onCompleted: {

    // invocation completed

}, disposeBag: disposeBag)
```

BSA.APICaller.requestSendOTPNumberForRegisterDevice

```
BSA.APICaller.requestSendOTPNumberForRegisterDevice(
                        clientKey:String,
                        userKey: String,
```

```
                        name: String,
                        sendType: BSA.APICaller.OTPSendType,
                        sendTarget: String,
                        bundleIdentifier: String,
                        onSuccess: @escaping (BSA.Enums.LocalAuthType) -> (),
                        onFailed: @escaping OnFailed,
                        onError: @escaping OnError,
                        onTotalLog: @escaping OnTotalLog? = nil,
                        onCompleted: @escaping onCompleted? = nil,
                        disposeBag: DisposeBag )
```

Parameter

Parameter	Type	Description
clientKey	String	Client's unique key
userKey	String	UserID
name	String	User name
sendType	BSA.APICaller.OTPSendType	Email, PhoneNumber
sendTarget	String	Address or number matching sendType
bundleIdentifier	String	App's bundleIdentifier
onSuccess	(BSA.Enums.LocalAuthType) -> ()	Callback called on success. It returns the last Local Authentication Type used by the user.
onFailed	OnFailed	Callback function called on API processing failure.
onError	OnError	Callback function called on error.
onTotalLog	OnTotalLog	Callback function to receive all logs generated during API calls.
onCompleted	onCompleted	Called at the end after all processes are completed.
disposeBag	DisposeBag	DisposeBag variable to manage RxSwift Disposable objects.

ii. OTP Number Validation

Enter the OTP number received by email/sms, call the API, and if successful, receive the **disposeToken** string. **disposeToken** is used to check whether the user is the correct user during Register New Device.

Example(BSA.APICaller.verifyOTPNumber

```
BSA.APICaller.verifyOTPNumber(sendType: BSA.APICaller.OTPSendType,
                              sendTarget: String,
                              otpNumber: "298336",
                              clientKey: "a1hrg2xv...",
onValidOTPNumber: { [weak self] disposeToken in
```

```
    // OTP Number Matches

    // Token Storage
    self?.data.disposeToken = disposeToken

}, onInvalidOTPNumber: {

    // OTP Number Doesn't Match

}, onFailed: { [weak self] rtCode, resultMessage in

    // Invocation Failure

}, onError: { [weak self] error in

    // Invocation Error

}, onTotalLog: { text in

    // Invocation Log

}, onCompleted: {

    // invocation completed

}, disposeBag: disposeBag)
```

```
BSA.APICaller.verifyOTPNumber(sendType: BSA.APICaller.OTPSendType,
                              sendTarget: String,
                               otpNumber: String,
                               clientKey: String,
                               onValidOTPNumber: @escaping (String) -> (),
                               onInvalidOTPNumber: @escaping () -> (),
                               onFailed: @escaping OnFailed,
                               onError: @escaping OnError,
                               onTotalLog: @escaping OnTotalLog? = nil,
                               onCompleted: @escaping onCompleted? = nil,
                               disposeBag: DisposeBag )
```

Parameter

Parameter	Type	Description
sendType	BSA.APICaller.OTPSendType	Platform to receive OTP number (Email, SMS)
sendTarget	String	Address for OTP number type
otpNumber	String	Received OTP number.
clientKey	String	Client's unique Key

Parameter	Type	Description
onValidOTPNumber	(String) -> ()	Callback called when OTP numbers match. The disposeToken value passed as a parameter is used in SignUp.
onInvalidOTPNumber	() -> ()	Callback called when OTP numbers don't match.
onFailed	OnFailed	Callback function called on API processing failure.
onError	OnError	Callback function called on error.
onTotalLog	OnTotalLog	Callback function to receive all logs generated during API calls.
onCompleted	onCompleted	Called at the end after all processes are completed.
disposeBag	DisposeBag	DisposeBag variable to manage RxSwift Disposable objects.

iii. Local Authentication

When called, it performs biometric and passcode authentication. For Register New Device, there is no saved **DomainState** for biometric authentication, so **checkDomainStateChanged** should be set to false.

Local Authentication must succeed to invoke **BSA.APICAller.requestSignUp**.
The callback called with **onResult** runs on a background thread, so UI operations should be done on the main thread.

Example

```
BSA.LocalAuth.startSimpleAuth(localAuthType: .BIOMETRIC,
                              reason: "Authentication is required to verify your identity",
                              checkDomainStateChanged: false,
                              usePasscodeWhenFailed: false,
onSuccess: {

    // onSuccess operates on a background thread, so UI-related tasks should be performed on the main thread.
    DispatchQueue.main.async {

        // Local Authentication Successful

    }

}, onError: { errorCode in

    // onError operates on a background thread, so UI-related tasks should be performed on the main thread.
    DispatchQueue.main.async {

        // Local Authentication Failed
```

```
        switch errorCode {
        case .DOMAIN_STATE_CHANGED:
            // Biometric authentication info has been changed
        case .BIOMETRIC_NOT_AVILABLE
        case .BIOMETRIC_NOT_SUPPORT_HARDWARE
        case .BIOMETRIC_LOCK_OUT:
            // Biometric authentication not available
        case .BIOMETRIC_NOT_ENROLLED_DEVICE:
            // Biometric authentication info is not on device
        case .PASSCODE_NOT_ENROLLED_DEVICE:
            //  Passcode not on device
        case .USER_CANCEL, .USER_FALLBACK:
            // User cancels authentication
        case .AUTH_FAILED:
            // Authentication failed
        default:
            // Other errors
        }

    }

})
```

BSA.LocalAuth.startSimpleAuth

```
BSA.LocalAuth.startSimpleAuth(
                        localAuthType: BSA.Enums.LocalAuthType,
                        reason: String,
                        checkDomainStateChanged: Bool,
                        usePasscodeWhenFailed:  Bool,
                        onSuccess: @escaping ()->(),
                        onError: @escaping (BSA.LocalAuthHelper.ErrorCode?)->())
```

Parameter

Parameter	Type	Description
localAuthType	BSA.Enums.LocalAuthType	Local authentication type
reason	String	Message displayed on OS during biometric authentication
checkDomainStateChanged	Bool	Determine whether to detect biometric changes
usePasscodeWhenFailed	Bool	Decide whether to use a passcode when biometric authentication fails
onSuccess	()->()	Local authentication processing result callback

Parameter	Type	Description
onError	(BSA.LocalAuthHelper.ErrorCode?)->()	Local authentication processing result callback

iv. Register New Device

existingUserData must be filled with complete data when making the call.

Example

```
BSA.APICaller.requestRegisterDevice(existingUserData: data.existingUserData!,
onSuccess: {
    // Successful Device Re-registration

}, onFailed: { [weak self] rtCode, resultMessage in

    // Invocation Failure

}, onError: { [weak self] error in

    // Invocation Error

}, onTotalLog: { text in

    // Invocation Log

}, onCompleted: {

    // invocation completed

}, disposeBag: disposeBag)
```

Parameter

```
BSA.APICaller.requestRegisterDevice(
                        existingUserData: BSA.SignUpData.ExistingUserData,
                        onSuccess: @escaping () -> (),
                        onFailed: @escaping OnFailed,
                        onError: @escaping OnError,
                        onTotalLog: @escaping OnTotalLog? = nil,
                        onCompleted: @escaping onCompleted? = nil,
                        disposeBag: DisposeBag )
```

Parameter	Type	Description

Parameter	Type	Description
existingUserData	BSA.SignUpData.ExistingUserData	Data required for Register New Device. Fill all fields and make the call.
onSuccess	() -> ()	Callback called on Register New Device success.
onFailed	OnFailed	Callback function called on API processing failure.
onError	OnError	Callback function called on error.
onTotalLog	OnTotalLog	Callback function to receive all logs generated during API calls.
onCompleted	onCompleted	Called at the end after all processes are completed.
disposeBag	DisposeBag	DisposeBag variable to manage RxSwift Disposable objects.

BSA.SignUpData.ExistingUserData

Name	Type	Description
appPackage	String	BundleIdentifier
clientKey	String	Client's unique key
appVersion	String	App version (e.g., 1.1.1)
token	String	FCM token value.
email	String	Email address.
phoneNumber	String	Phone number.
userKey	String	UserID
name	String	User name
authType	BSA.Enums.LocalAuthType	Local Authentication Type (NONE, BIOMETRIC, PASSCODE)
os	BSA.Enums.OSType	os type (fixed to iOS)
disposeToken	String	Token received upon OTP authentication success.
otpType	BSA.Enums.OTPType	Type of OTP number received (email, SMS).

If Register New Device is successful, you can retrieve information using the following functions.
BSA.loadTOTPSecretKey
BSA.loadDeviceID
BSA.loadFCMToken
BSA.loadUserKey
BSA.loadLocalAuthType

Authentication

i. Access Token Check

API : BSA.loadAccessToken

```
if let token = BSA.loadAccessToken() {
    // While Holding Access Token

}
else {
    // No Access Token

}
```

Description

- If no `Access Token` is stored, it returns nil.
- The `Access Token` is stored internally in the SDK when the basic authentication logic is successfully completed through `AutoAuthenticator`.
- Most user-related functions such as query, authentication type change, and withdrawal require the `Access Token`.

ii. AutoAuthenticator

API : BSA.AutoAuthenticator()

```
// authenticator
let autoAuthenticator = BSA.AutoAuthenticator()

// parameters
let qrId: String? = nil
let pushData: BSA.FCM.PushData?  = nil
let clientKey = "a1hrg2xv..."

guard let userKey = BSA.loadUserKey() else {
    return false
}

guard let deviceId = BSA.loadDeviceID() else {
    return false
}
```

```
let otpAuth = false

let sendPush = false
let localAuthType = BSA.loadLocalAuthType() ?? .BIOMETRIC
let usePasscodeWhenFailed = false

// setData
autoAuthenticator.setData(qrId: qrId,
                          pushData: pushData,
                          clientKey: clientKey,
                          userKey: userKey,
                          deviceId: deviceId,
                          otpAuth: otpAuth,
                          sendPushMessage: sendPush,
                          localAuthType: localAuthType,
                          usePasscodeWhenFailed: usePasscodeWhenFailed)
// setEvent
.setEvent(onProcessing: { [weak self] processDescription in

    // onProcessing operates in the background thread, so processing such as UI
must be done in the main thread.
    DispatchQueue.main.async {

        // Process Verification
        var processName = ""
        switch(processDescription) {
            case .StartAuth:
                processName = "Start Authentication"

                .
                .
                .
        }

        self?.labelProcess.text = processName

    }

}, onSuccess: { [weak self] in

    // Authentication Success

}, onCancel: { [weak self] in

    // Authentication Canceled

}, onFailed: { [weak self] rtCode, resultMessage in

    // Authentication Failed

}, onError: { [weak self] error in

    // Invocation Error
```

```
}, onTotalLog: { [weak self] totalLog in

    // Invocation Log

}).start() // Start Authentication
```

Parameters

setData

```
autoAuthenticator.setData(qrId: String?,
                          pushData: BSA.FCM.PushData?,
                          clientKey: String,
                          userKey: String,
                          deviceId: String,
                          otpAuth: Bool,
                          sendPushMessage: Bool,
                          localAuthType: BSA.Enums.LocalAuthType,
                          usePasscodeWhenFailed: Bool) -> AutoAuthenticator
```

Parameter	Type	Description
qrId	String?	QR recognition result. If it's not QR authentication, it's nil.
pushData	BSA.FCM.PushData?	Authentication data received via push. If it's not Push Message Authentication, it's nil.
clientKey	String	Client's unique key
userKey	String	UserID
deviceId	String	UUID processed by the SDK.
otpAuth	Bool	Check if it's OTP authentication.
sendPushMessage	Bool	Check if a push message should be sent.
LocalAuthType	BSA.Enums.LocalAuthType	Local Authentication Type
userPasscodeWhenFailed	Bool	Determine whether to use passcode in case of biometric authentication failure.

setEvent

```
autoAuthenticator.setEvent(
            onProcessing: @escaping (BSA.Enums.ProcessDescription) -> (),
            onSuccess: @escaping () -> (),
            onCancel: @escaping () -> (),
            onFailed: @escaping BSA.APICaller.OnFailed,
            onError: @escaping BSA.APICaller.OnError,
            onTotalLog: @escaping BSA.APICaller.OnTotalLog) -> AutoAuthenticator
```

Parameter	Type	Description
onProcessing	(BSA.Enums.ProcessDescription) -> ()	Name of the currently processing process.
onSuccess	() -> ()	Called on success.
onCancel	() -> ()	Called on Cancel.
onFailed	OnFailed	Callback function called on API processing failure.
onError	OnError	Callback function called on error.
onTotalLog	OnTotalLog	Callback function to receive all logs generated during API calls.

start

```
autoAuthenticator.start()
```

Description

- When using `AutoAuthenticator`, you must call `setData` and `setEvent` before calling`start`.
- After calling `start`, it automatically performs all authentication processes `(authentication start -> blockchain node verification -> local authentication -> authentication completion)`.
    - `onProcessing` is called with the name of the authentication process when that authentication process starts.
- If authentication is successful, the authentication token issued by the authentication is stored internally in the SDK (It's Called an AccessToken).
    - Caution: If you proceed with QR or push authentication, a token will not be issued.
    - Depending on the `removeToken` parameter when calling `BSA.initialize`, this token can be deleted or preserved.
- For the `otpAuth` parameter in `setData`, you should put the `otpAuth` value from push data.

    - If it's not Push Message Authentication, it's `false`.
- Select and use the `clientKey` appropriately internally according to QR, Push, and Basic Authentication.

ii-i. Basic Authentication

- When requesting authentication for the purpose of obtaining `Access Token` to use other APIs.
- If the parameters of `setData` are set as below, it's Basic Authentication.

```
// parameters
let qrId: String? = nil
let pushData: BSA.FCM.PushData?  = nil
let otpAuth = false
```

ii-ii. QR Authentication

```
var qrHelper = BSA.QRHelper()

var cameraParent: UIView = "Parent View with Camera Preview"

// Check Camera Permission
qrHelper.checkCameraPermission(onGranted: { [weak self] in

    // Start QR Parsing Logic
    self?.qrHelper.startQR(cameraParent: cameraParent,
                        onPreviewLayerAdded: { cameraParent, previewLayer in

        // UI handling

    }, onSuccess: { [weak self] qrData in
        // QR Recognition Successful
        self?.data.qrData = qrData

    }, onFailed: { [weak self] qrError in
         // QR Recognition Failed

    })

}, onDenied: {
    // No Permission

})
```

- If you want to proceed with QR Authentication, obtain QRData using `BSA.QRHelper` and then set `qrId: String?` in `AuthAuthenticator`'s `setData`.

If a QR process already exists

```
var qrHelper = BSA.QRHelper()

let qrData = qrHelper.parseData(code: "String that parses the QR code")

if qrData.qrResult { //If the authentication QR code for BSA is correct

    // TODO: action on success

} else { //When the QR code is not in the correct form (when it is not a related QR code)

    // TODO: Action on failure

}
```

- If there is a QR process, you can call `qrHelper.parseData()` on the string that parsed the QR and use the resulting value.

ii-iii. Push Message Authentication

- If you want to proceed with Push Message Authentication, set the `pushData` obtained through the callback of `BSA.FCM.registerPushNotificationProcessor` as `pushData: BSA.FCM.PushData?` in`AuthAuthenticator`'s `setData`.
    - Refer to the `FCM Setup and Configuration Guide` for details.

ii-iv. OTP Authentication

- There are two methods for OTP authentication:
    - 1. `Press OTP authentication after entering the UserID in a different platform(web)` and receive push first.
    - 2. `Press OTP authentication without entering the UserID in a different platform(web)` and receive OTP number first.

- For the first method, after successful Push Message Authentication, receive the OTP number and enter it in the requesting platform.
- For the second method, first call `BSA.APICaller.issuanceOTPNumber` to receive the OTP number, then enter it in the requesting platform and proceed with Push Message Authentication.

Issuance OTP Number

API : BSA.APICaller.issuanceOTPNumber

```
BSA.APICaller.issuanceOTPNumber(onSuccess: { [weak self] otpNumber in

    // Issuance Successful

}, onTokenExpired: { [weak self] rtCode, resultMessage in

    // Token Expired

}, onFailed: { rtCode, resultMessage in

    // Invocation Failure

}, onError: { error in

    // Invocation Error

}, onTotalLog: { text in

    // Invocation Log

}, onCompleted: {

    // invocation completed

}, disposeBag: disposeBag)
```

OTP Issuance Cancellation

API : BSA.APICaller.IssuanceCancelForOTPNumber

```
let otpNumber = "226339" // OTP Number to Cancel

BSA.APICaller.IssuanceCancelForOTPNumber(otpNumber: otpNumber,
onSuccess: { [weak self] otpNumber in

    // Cancellation of Issuance Successful
```

```
}, onTokenExpired: { [weak self] rtCode, resultMessage in

    // Token Expired

}, onFailed: { rtCode, resultMessage in

    // Invocation Failure

}, onError: { error in

    // Invocation Error

}, onTotalLog: { text in

    // Invocation Log

}, onCompleted: {

    // invocation completed

}, disposeBag: disposeBag)
```

ii-v. TOTP Authentication

- TOTP is used as an alternative for authentication when network connectivity is not available.
- SignUp or Register New Device must proceed successfully.
    - Use `BSA.loadTOTPSecretKey()` issued during SignUp or Register New Device to internally generate TOTP numbers.
-

```
let result = BSA.TOTP.generateTOTP()

if let totpNumber = result.number {
    // Successful TOTP Number Generation

}
else {
    // If result.number is nil, it's an error
    print(result.error)

}
```

iii. Authentication Cancellation

API : BSA.APICaller.requestCancelAuth

```
guard let deviceId = BSA.loadDeviceID() else {
    return
}

guard let userKey = BSA.loadUserKey() else {
    return
}

BSA.APICaller.requestCancelAuth(qrId: nil,
                                clientKey: "a1hrg2xv...",
                                deviceId: deviceId,
                                userKey: userKey,
                                sendPushMessage: false,
onSuccess: {
    // Successful Invocation


}, onTokenExpired: { [weak self] rtCode, resultMessage in

    // Token Expired

}, onFailed: { [weak self] rtCode, resultMessage in

    // Invocation Failure

}, onError: { [weak self] error in

    // Invocation Error

}, onTotalLog: { text in

    // Invocation Log

}, onCompleted: {

    // invocation completed

}, disposeBag: disposeBag)
```

Parameters

```
BSA.APICaller.requestCancelAuth(qrId: String?,
                clientKey: String,
```

```
                deviceId: String,
                userKey: String,
                sendPushMessage: Bool,
                onSuccess: @escaping () -> (),
                onTokenExpired: @escaping BSA.APICaller.OnTokenExpired,
                onFailed: @escaping OnFailed,
                onError: @escaping OnError,
                onTotalLog: @escaping OnTotalLog? = nil,
                onCompleted: @escaping onCompleted? = nil,
                disposeBag: DisposeBag)
```

Parameter	Type	Description
qrId	String?	QR recognition result. If it's not QR authentication, it's nil.
clientKey	String	Client's unique key
deviceId	String	UUID processed by the SDK.
userKey	String	UserID
sendPushMessage	Bool	Check if a push message should be sent.
onSuccess	() -> ()	Called on success.
onTokenExpired	onTokenExpired	Function called when token expires. Same format as onFailed
onFailed	OnFailed	Callback function called on API processing failure.
onError	OnError	Callback function called on error.
onTotalLog	OnTotalLog	Callback function to receive all logs generated during API calls.
onCompleted	onCompleted	Called at the end after all processes are completed.
disposeBag	DisposeBag	DisposeBag variable to manage RxSwift Disposable objects.

Description

- When cancellation is needed during authentication, you can request it.
    - When this API is called, the operations of `AutoAuthenticator` are canceled.
- When canceling during QR Authentication request, `qrId` is required.

iv. Biometric Authentication Re-registration

- If biometric authentication information (FaceID, TouchID) changes after SignUp or Register New Device, it will fail in Auth-Logic.
- If it has changed:
  - `BSA.APICaller.requestOTPNumberSend`
  - `BSA.APICaller.verifyOTPNumber`
  - `BSA.LocalAuth.startSimpleAuth`
- If performed in sequence, the biometric authentication information will be updated.
  - Detailed API information can be found in the SignUp Guide.
  - Biometric authentication information itself can be resolved by calling `BSA.LocalAuth.startSimpleAuth`, but for security, OTP number must be requested and authenticated.

User Information Modification

i. Change Local Authentication Type

API : BSA.APICaller.changeLocalAuthType

```
BSA.APICaller.changeLocalAuthType(authType: .BIOMETRIC, onSuccess: {

    // Modification Successful

    // Check Modified Values
    let localAuthType = BSA.loadLocalAuthType() ?? .UNKNOWN


}, onTokenExpired: { [weak self] rtCode, resultMessage in

    // Token Expired

}, onFailed: { [weak self] rtCode, resultMessage in

    // Invocation Failure

}, onError: { [weak self] error in

    // Invocation Error

}, onTotalLog: { text in

    // Invocation Log

}, onCompleted: {

    // invocation completed

}, disposeBag: disposeBag)
```

Parameters

```
BSA.APICaller.changeLocalAuthType(authType: BSA.Enums.LocalAuthType,
                  onSuccess: @escaping () -> (),
                  onTokenExpired: @escaping BSA.APICaller.OnTokenExpired,
                  onFailed: @escaping OnFailed,
                  onError: @escaping OnError,
                  onTotalLog: @escaping OnTotalLog? = nil,
                  onCompleted: @escaping onCompleted? = nil,
                  disposeBag: DisposeBag )
```

Parameter	Type	Description
authType	BSA.Enums.LocalAuthType	Local Authentication Type to be changed.
onSuccess	() -> ()	Called on success
onTokenExpired	onTokenExpired	Function called when token expires. Same format as onFailed
onFailed	OnFailed	Callback function called on API processing failure.
onError	OnError	Callback function called on error.
onTotalLog	OnTotalLog	Callback function to receive all logs generated during API calls.
onCompleted	onCompleted	Called at the end after all processes are completed.
disposeBag	DisposeBag	DisposeBag variable to manage RxSwift Disposable objects.

Description

- Change the user's default Local Authentication Type.
- The user's default Local Authentication Type can be determined by calling `BSA.loadLocalAuthType()` after invoking this function.

ii. Unregister Device

API : BSA.APICaller.requestDeregisterDevice

```
guard let userKey = BSA.loadUserKey() else {
    return
}

BSA.APICaller.requestDeregisterDevice(userKey: userKey,
onSuccess: {
    // Successful Invocation


}, onTokenExpired: { [weak self] rtCode, resultMessage in

    // Token Expired

}, onFailed: { [weak self] rtCode, resultMessage in
```

```
    // Invocation Failure
}, onError: { [weak self] error in
    // Invocation Error
}, onTotalLog: { text in
    // Invocation Log
}, onCompleted: {
    // invocation completed
}, disposeBag: disposeBag)
```

Parameters

```
BSA.APICaller.requestDeregisterDevice(userKey: String,
                  onSuccess: @escaping () -> (),
                  onTokenExpired: @escaping BSA.APICaller.OnTokenExpired,
                  onFailed: @escaping OnFailed,
                  onError: @escaping OnError,
                  onTotalLog: @escaping OnTotalLog? = nil,
                  onCompleted: @escaping onCompleted? = nil,
                  disposeBag: DisposeBag )
```

Parameter	Type	Description
userKey	String	UserID
onSuccess	() -> ()	Called on success
onTokenExpired	onTokenExpired	Function called when token expires. Same format as onFailed
onFailed	OnFailed	Callback function called on API processing failure.
onError	OnError	Callback function called on error.
onTotalLog	OnTotalLog	Callback function to receive all logs generated during API calls.
onCompleted	onCompleted	Called at the end after all processes are completed.
disposeBag	DisposeBag	DisposeBag variable to manage RxSwift Disposable objects.

Description

- This should be called when Access Token is issued.
- When Unregister Device is performed, all user data stored is deleted.

- TOTP Key
- DeviceID
- AccessToken
- LocalAuthType
- FCMToken
- UserKey

iii. Delete Account

API : BSA.APICaller.requestDeleteAccount

```
BSA.APICaller.requestDeleteAccount(onSuccess: {
    // Successful Invocation


}, onTokenExpired: { [weak self] rtCode, resultMessage in

    // Token Expired

}, onFailed: { [weak self] rtCode, resultMessage in

    // Invocation Failure

}, onError: { [weak self] error in

    // Invocation Error

}, onTotalLog: { text in

    // Invocation Log

}, onCompleted: {

    // invocation completed

}, disposeBag: disposeBag)
```

Parameters

```
BSA.APICaller.requestDeleteAccount(onSuccess: @escaping () -> (),
                onTokenExpired: @escaping BSA.APICaller.OnTokenExpired,
                onFailed: @escaping OnFailed,
                onError: @escaping OnError,
```

```
                onTotalLog: @escaping OnTotalLog? = nil,
                onCompleted: @escaping onCompleted? = nil,
                disposeBag: DisposeBag
```

Parameter	Type	Description
onSuccess	() -> ()	Called on success.
onTokenExpired	onTokenExpired	Function called when token expires. Same format as onFailed
onFailed	OnFailed	Callback function called on API processing failure.
onError	OnError	Callback function called on error.
onTotalLog	OnTotalLog	Callback function to receive all logs generated during API calls.
onCompleted	onCompleted	Called at the end after all processes are completed.
disposeBag	DisposeBag	DisposeBag variable to manage RxSwift Disposable objects.

Description

- This should be called when Access Token is issued.
- When Delete Account is performed, all user data stored is deleted.
    - TOTP Key
    - DeviceID
    - AccessToken
    - LocalAuthType
    - FCMToken
    - UserKey

Other APIs (Need for Access Token)

The following APIs require a valid Access Token obtained through authentication to be invoked successfully.

i. Authentication History Query

API : BSA.APICaller.retrieveAuthHistory

```
BSA.APICaller.retrieveAuthHistory(pageIndex:0, size: 10,
                                  onSuccess: { arrayData, pageData in

    // Successful Invocation


}, onTokenExpired: { [weak self] rtCode, resultMessage in

    // □Token Expired

}, onFailed: { [weak self] rtCode, resultMessage in

    // Invocation Failure

}, onError: { [weak self] error in

    // Invocation Error

}, onTotalLog: { text in

    // Invocation Log

}, onCompleted: {

    // invocation completed

}, disposeBag: disposeBag)
```

Parameters

```
BSA.APICaller.retrieveAuthHistory(
                pageIndex: Int?,
                size: Int?,
                sort: String = "REG_DT,DESC",
                onSuccess: @escaping ([BSA.JSONData.AuthHistoryData],
BSA.JSONData.AuthHistoryPage) -> (),
```

```
                onTokenExpired: @escaping BSA.APICaller.OnTokenExpired,
                onFailed: @escaping OnFailed,
                onError: @escaping OnError,
                onTotalLog: @escaping OnTotalLog? = nil,
                onCompleted: @escaping onCompleted? = nil,
                disposeBag: DisposeBag )
```

Parameter	Type	Description
pageIndex	Int?	Index of page to load
size	Int?	Number of elements to load at once
sort	String	sorting method. "REG_DT,DESC" = Sort by newest
onSuccess	([BSA.JSONData.AuthHistoryData], BSA.JSONData.AuthHistoryPage) -> ()	Callback called on success. It returns an array of authentication history data and page data.
onTokenExpired	onTokenExpired	Function called when token expires. Same format as onFailed
onFailed	OnFailed	Callback function called on API processing failure.
onError	OnError	Callback function called on error.
onTotalLog	OnTotalLog	Callback function to receive all logs generated during API calls.
onCompleted	onCompleted	Called at the end after all processes are completed.
disposeBag	DisposeBag	DisposeBag variable to manage RxSwift Disposable objects.

Description

- Information about authentication history is contained in `[BSA.JSONData.AuthHistoryData]` returned on success.
- `BSA.JSONData.AuthHistoryPage` contains page information such as the maximum number of entries.

Related Data

BSA.JSONData.AuthHistoryData

Name	Type	Description
seq	Int	Sequence.
userKey	String	UserID
clientKey	String	Client's unique key
clientName	String	□Client name
status	BSA.Enums.AuthHistoryStatus	Status (success = "CMMASC001", failed = "CMMASC002", canceled = "CMMASC003", timeout = "CMMASC004")
platform	BSA.Enums.AuthPlatform	Platform (iOS = CMMAPF002)
content	String	Content.
connectIp	String	Access IP.
regDt	String	Access □date. (Default)
regDt_ko	String	Access date. (Korean format)
regDt_en	String	Access date. (English format)

BSA.JSONData.AuthHistoryPage

Name	Type	Description
first	Bool	Check if the current data is on the first page.
last	Bool	Check if the current data is on the last page.
currentPage	Int	Current page number.
pageSize	Int	Number of data entries per page.
totalElements	Int	Total number of data entries.
totalPages	Int	Total number of pages calculated based on (pageSize).

ii. Linked Site List Query

API : BSA.APICaller.retrieveLinkedSiteList

```
BSA.APICaller.retrieveLinkedSiteList(onSuccess: { arrayData in
```

```
    // Successful Invocation

}, onTokenExpired: { [weak self] rtCode, resultMessage in

    // □Token Expired

}, onFailed: { [weak self] rtCode, resultMessage in

    // Invocation Failure

}, onError: { [weak self] error in

    // Invocation Error

}, onTotalLog: { text in

    // Invocation Log

}, onCompleted: {

    // invocation completed

}, disposeBag: disposeBag)
```

Parameters

```
BSA.APICaller.retrieveLinkedSiteList(
            onSuccess: @escaping ([BSA.JSONData.LinkedSiteListData]) -> (),
            onTokenExpired: @escaping BSA.APICaller.OnTokenExpired,
            onFailed: @escaping OnFailed,
            onError: @escaping OnError,
            onTotalLog: @escaping OnTotalLog? = nil,
            onCompleted: @escaping onCompleted? = nil,
            disposeBag: DisposeBag )
```

Parameter	Type	Description
onSuccess	([BSA.JSONData.LinkedSiteListData]) -> ()	Callback called on success. It returns information about linked sites.
onTokenExpired	onTokenExpired	Function called when token expires. Same format as onFailed
onFailed	OnFailed	Callback function called on API processing failure.
onError	OnError	Callback function called on error.

Parameter	Type	Description
onTotalLog	OnTotalLog	Callback function to receive all logs generated during API calls.
onCompleted	onCompleted	Called at the end after all processes are completed.
disposeBag	DisposeBag	DisposeBag variable to manage RxSwift Disposable objects.

Description

- Information about linked sites is contained in `[BSA.JSONData.LinkedSiteListData]` returned on success.
- It retrieves the list of sites currently linked by the user.

Related Data

BSA.JSONData.LinkedSiteListData

Name	Type	Description
seq	Int	Sequence.
clientKey	String	Client's unique key
clientName	String	Client name
userStatus	String	Status (normal = "CMMMST001", wait = "CMMMST002", block = "CMMMST003", deleted = "CMMMST004", withdraw = "CMMMST005", reject = "CMMMST006", none = "")
clientExplain	String?	Site Description
siteUrl	String	Site Address
interlock	Bool	Link Status
verifyType	String	Authentication Type (if there is a value, it requires an account for that site)

iii. Complete Site List Query

API : BSA.APICaller.retrieveSiteList

```
BSA.APICaller.retrieveSiteList(clientName: "",
onSuccess: { arrayData in
    // Successful Invocation


}, onTokenExpired: { [weak self] rtCode, resultMessage in

    // □Token Expired

}, onFailed: { [weak self] rtCode, resultMessage in

    // Invocation Failure

}, onError: { [weak self] error in

    // Invocation Error

}, onTotalLog: { text in

    // Invocation Log

}, onCompleted: {

    // invocation completed

}, disposeBag: disposeBag)
```

Parameters

```
BSA.APICaller.retrieveSiteList(
                clientName: String?,
                onSuccess: @escaping ([BSA.JSONData.SiteListData]) -> (),
                onTokenExpired: @escaping BSA.APICaller.OnTokenExpired,
                onFailed: @escaping OnFailed,
                onError: @escaping OnError,
                onTotalLog: @escaping OnTotalLog? = nil,
                onCompleted: @escaping onCompleted? = nil,
                disposeBag: DisposeBag )
```

Parameter	Type	Description
clientName	String	Site name to search for.
onSuccess	([BSA.JSONData.SiteListData]) -> ()	Callback called on success. It returns an array of site information.

Parameter	Type	Description
onTokenExpired	onTokenExpired	Function called when token expires. Same format as onFailed
onFailed	OnFailed	Callback function called on API processing failure.
onError	OnError	Callback function called on error.
onTotalLog	OnTotalLog	Callback function to receive all logs generated during API calls.
onCompleted	onCompleted	Called at the end after all processes are completed.
disposeBag	DisposeBag	DisposeBag variable to manage RxSwift Disposable objects.

Description

- Information about sites is contained in `[BSA.JSONData.SiteListData]` returned on success.
- If you input a space ("") for the `clientName` parameter, it returns information about all sites.
- It returns information about sites with site names containing the string you provided in the `clientName` parameter.

Related Data

BSA.JSONData.SiteListData

Name	Type	Description
seq	Int	Sequence.
clientKey	String	Client's unique key
clientName	String	Client name
siteUrl	String	Site Address
interlock	Bool	Link Status
verifyType	String	Authentication Type (if there is a value, it requires an account for that site)

iv. Site Account Authentication

API : BSA.APICaller.requestVerifySiteAccount

```
BSA.APICaller.requestVerifySiteAccount(siteClientKey: siteClientKey,
                                       siteID: siteID,
                                       sitePassword: sitePassword,
onSuccess: { token in
    // Successful Invocation


}, onTokenExpired: { [weak self] rtCode, resultMessage in

    // □Token Expired

}, onFailed: { [weak self] rtCode, resultMessage in

    // Invocation Failure

}, onError: { [weak self] error in

    // Invocation Error

}, onTotalLog: { text in

    // Invocation Log

}, onCompleted: {

    // invocation completed

}, disposeBag: disposeBag)
```

Parameters

```
BSA.APICaller.requestVerifySiteAccount(siteClientKey: String,
               siteID: String,
               sitePassword: String,
               onSuccess: @escaping (String) -> (),
               onTokenExpired: @escaping BSA.APICaller.OnTokenExpired,
               onFailed: @escaping OnFailed,
               onError: @escaping OnError,
               onTotalLog: @escaping OnTotalLog? = nil,
               onCompleted: @escaping onCompleted? = nil,
               disposeBag: DisposeBag )
```

Parameter	Type	Description
siteClientKey	String	Client's unique key of the site to authenticate.
siteID	String	Account ID of the site to authenticate.
sitePassword	String	Account password of the site to authenticate.
onSuccess	(String) -> ()	Callback called on success. It returns a Verification Token.
onTokenExpired	onTokenExpired	Function called when token expires. Same format as onFailed
onFailed	OnFailed	Callback function called on API processing failure.
onError	OnError	Callback function called on error.
onTotalLog	OnTotalLog	Callback function to receive all logs generated during API calls.
onCompleted	onCompleted	Called at the end after all processes are completed.
disposeBag	DisposeBag	DisposeBag variable to manage RxSwift Disposable objects.

Description

- If the site you want to link has a `verifyType` value among the `BSA.JSONData.SiteListData` obtained from `BSA.APICaller.retrieveSiteList`, you need to confirm if the account is valid by calling `BSA.APICaller.requestVerifySiteAccount` with any value for the `verifyType`.
- If the account information for the site is valid and the callback is successful, you should proceed with `BSA.APICaller.requestLinkSite` using the Verification Token returned.

v. Site Linking Request

API : BSA.APICaller.requestLinkSite

```
BSA.APICaller.requestLinkSite(siteClientKey: siteClientKey,
                              siteToken: siteToken,
onSuccess: {
    // Successful Invocation


}, onTokenExpired: { [weak self] rtCode, resultMessage in

    // □Token Expired

}, onFailed: { [weak self] rtCode, resultMessage in
```

```
    // Invocation Failure
}, onError: { [weak self] error in
    // Invocation Error
}, onTotalLog: { text in
    // Invocation Log
}, onCompleted: {
    // invocation completed
}, disposeBag: disposeBag)
```

Parameters

```
BSA.APICaller.requestLinkSite(siteClientKey: String,
              siteToken: String?,
              onSuccess: @escaping () -> (),
              onTokenExpired: @escaping BSA.APICaller.OnTokenExpired,
              onFailed: @escaping OnFailed,
              onError: @escaping OnError,
              onTotalLog: @escaping OnTotalLog? = nil,
              onCompleted: @escaping onCompleted? = nil,
              disposeBag: DisposeBag )
```

Parameter	Type	Description
siteClientKey	String	Client's unique key of the site to link.
siteToken	String	Token obtained after account verification. If you do not perform account verification, use nil.
onSuccess	(String) -> ()	Called on success.
onTokenExpired	onTokenExpired	Function called when token expires. Same format as onFailed
onFailed	OnFailed	Callback function called on API processing failure.
onError	OnError	Callback function called on error.
onTotalLog	OnTotalLog	Callback function to receive all logs generated during API calls.
onCompleted	onCompleted	Called at the end after all processes are completed.
disposeBag	DisposeBag	DisposeBag variable to manage RxSwift Disposable objects.

Description

- If the site you want to link has a `verifyType` value, you need to confirm the account for that site first.
  - You should first proceed with `BSA.APICaller.requestVerifySiteAccount` to obtain the `siteToken`.
- If the site you want to link does not have a `verifyType` value, you can use nil for `siteToken`.

vi. Site Unlinking Request

API : BSA.APICaller.requestUnlinkSite

```
BSA.APICaller.requestUnlinkSite(siteClientKey: siteClientKey,
onSuccess: {
    // Successful Invocation


}, onTokenExpired: { [weak self] rtCode, resultMessage in

    // □Token Expired

}, onFailed: { [weak self] rtCode, resultMessage in

    // Invocation Failure

}, onError: { [weak self] error in

    // Invocation Error

}, onTotalLog: { text in

    // Invocation Log

}, onCompleted: {

    // invocation completed

}, disposeBag: disposeBag)
```

Parameters

```
BSA.APICaller.requestUnlinkSite(siteClientKey: String,
                onSuccess: @escaping () -> (),
                onTokenExpired: @escaping BSA.APICaller.OnTokenExpired,
```

```
                onFailed: @escaping OnFailed,
                onError: @escaping OnError,
                onTotalLog: @escaping OnTotalLog,
                disposeBag: DisposeBag )
```

Parameter	Type	Description
siteClientKey	String	Client's unique key of the site to unlink.
onSuccess	() -> ()	Called on success.
onTokenExpired	onTokenExpired	Function called when token expires. Same format as onFailed
onFailed	OnFailed	Callback function called on API processing failure.
onError	OnError	Callback function called on error.
onTotalLog	OnTotalLog	Callback function to receive all logs generated during API calls.
onCompleted	onCompleted	Called at the end after all processes are completed.
disposeBag	DisposeBag	DisposeBag variable to manage RxSwift Disposable objects.

Description

- This request will unlink the site matching `siteClientKey`.

Other APIs (No need for Access Token)

i. Terms and Conditions

API : BSA.APICaller.getAgreements

```
BSA.APICaller.getAgreements(languageCode: .korean,
                            clientKey: "a1hrg2xv...",
onSuccess: { [weak self] arrayData in

    // Successful Invocation
    self?.data.arrayAgreementData = arrayData


}, onFailed: { [weak self] rtCode, resultMessage in

    // Invocation Failure

}, onError: { [weak self] error in

    // Invocation Error

}, onTotalLog: { text in

    // Invocation Log

}, onCompleted: {

    // invocation completed

}, disposeBag: disposeBag)
```

Parameters

```
BSA.APICaller.getAgreements(languageCode: BSA.Enums.AgreementsLanguageCode,
                clientKey: String,
                onSuccess: @escaping ([BSA.JSONData.AgreementData]) -> (),
                onFailed: @escaping OnFailed,
                onError: @escaping OnError,
                onTotalLog: @escaping OnTotalLog? = nil,
                onCompleted: @escaping onCompleted? = nil,
                disposeBag: DisposeBag )
```

Parameter	Type	Description
languageCode	BSA.Enums.AgreementsLanguageCode	Language of the terms and conditions.
clientKey	String	Client's unique key

Parameter	Type	Description
onSuccess	([BSA.Enums.AgreementData]) -> ()	Callback called on success. It returns an array of data that allows you to view detailed terms and conditions.
onFailed	OnFailed	Callback function called on API processing failure.
onError	OnError	Callback function called on error.
onTotalLog	OnTotalLog	Callback function to receive all logs generated during API calls.
onCompleted	onCompleted	Called at the end after all processes are completed.
disposeBag	DisposeBag	DisposeBag variable to manage RxSwift Disposable objects.

Description

- You can get approximate information about the existing terms and conditions, and if you want detailed information about a specific term and condition, you need to call `BSA.APICaller.getAgreementDetail` using the seq from `BSA.Enums.AgreementData`.

Related Data

BSA.Enums.AgreementData

Name	Type	Description
clientKey	String	Client's unique key
lang	String	Language.
seq	Int	Sequence.
title	String	Title.
type	String	Type of terms and conditions.

i-i. Detailed Terms and Conditions

API : BSA.APICaller.getAgreementDetail

```
BSA.APICaller.getAgreementDetail(seq: seq,
                                 clientKey: "a1hrg2xv...",
onSuccess: { [weak self] agreementDetailData in

    // Successful Invocation

}, onFailed: { [weak self] rtCode, resultMessage in

    // Invocation Failure

}, onError: { [weak self] error in

    // Invocation Error

}, onTotalLog: { text in

    // Invocation Log

}, onCompleted: {

    // invocation completed

}, disposeBag: disposeBag)
```

Parameters

```
BSA.APICaller.getAgreementDetail(seq: Int,
              clientKey: String,
              onSuccess: @escaping (BSA.JSONData.AgreementDetailData) -> (),
              onFailed: @escaping OnFailed,
              onError: @escaping OnError,
              onTotalLog: @escaping OnTotalLog? = nil,
              onCompleted: @escaping onCompleted? = nil,
              disposeBag: DisposeBag )
```

Parameter	Type	Description
seq	Int	BSA.Enums.AgreementData's seq value
clientKey	String	Client's unique key
onSuccess	(BSA.Enums.AgreementDetailData) -> ()	Callback called on success. It contains detailed information about the terms and conditions.
onFailed	OnFailed	Callback function called on API processing failure.
onError	OnError	Callback function called on error.

Parameter	Type	Description
onTotalLog	OnTotalLog	Callback function to receive all logs generated during API calls.
onCompleted	onCompleted	Called at the end after all processes are completed.
disposeBag	DisposeBag	DisposeBag variable to manage RxSwift Disposable objects.

Description

- Retrieve detailed information about the terms and conditions that match the seq.

Related Data

BSA.Enums.AgreementDetailData

Name	Type	Description
regDt	String	Registration date.
lang	String	Language.
clientKey	String	Client's unique key
clientName	String	Client name
regUserKey	String	Registrant's ID.
seq	Int	Sequence.
title	String	Title.
type	String	Type of terms and conditions.
content	String	Terms and conditions text.

ii. Notice

API : BSA.APICaller.retrieveNotice

```
BSA.APICaller.retrieveNotice(clientKey: "a1hrg2xv...",
                             pageIndex: pageIndex,
                             pageSize: pageSize,
 onSuccess: { arrayData, pageData in

    // Successful Invocation

}, onFailed: { [weak self] rtCode, resultMessage in

    // Invocation Failure

}, onError: { [weak self] error in

    // Invocation Error

}, onTotalLog: { text in

    // Invocation Log

}, onCompleted: {

    // invocation completed

}, disposeBag: disposeBag)
```

Parameters

```
BSA.APICaller.retrieveNotice(clientKey: String,
                             pageIndex: Int,
                             pageSize: Int,
onSuccess: @escaping ([BSA.JSONData.NoticeData], BSA.JSONData.NoticePage) -> (),
                             onFailed: @escaping OnFailed,
                             onError: @escaping OnError,
                             onTotalLog: @escaping OnTotalLog? = nil,
                             onCompleted: @escaping onCompleted? = nil,
                             disposeBag: DisposeBag )
```

Parameter	Type	Description
clientKey	String	Client's unique key
pageIndex	Int	Page number to be queried. The value for the first page is 0.
pageSize	Int	Number of Notice entries per page.
onSuccess	([BSA.JSONData.NoticeData], BSA.JSONData.NoticePage) -> ()	Callback called on success. It returns an array of Notice data and page information.

Parameter	Type	Description
onFailed	OnFailed	Callback function called on API processing failure.
onError	OnError	Callback function called on error.
onTotalLog	OnTotalLog	Callback function to receive all logs generated during API calls.
onCompleted	onCompleted	Called at the end after all processes are completed.
disposeBag	DisposeBag	DisposeBag variable to manage RxSwift Disposable objects.

Description

- The information about Notice is contained in `[BSA.JSONData.NoticeData]` returned on success.
- `BSA.JSONData.NoticePage` contains page information such as the maximum number of entries per page.

Related Data

BSA.JSONData.NoticeData

Name	Type	Description
seq	Int	Sequence.
version	String	App's Version
patchType	String	OS Type
title	String	Title.
regUserName	String	Registrant's ID.
content	String	Content.
deplyDt	String	Distribution date.
regDt	String	Registration date.

BSA.JSONData.NoticePage

Name	Type	Description
first	Bool	Check if the current data is on the first page.

Name	Type	Description
last	Bool	Check if the current data is on the last page.
currentPage	Int	Current page number.
pageSize	Int	Number of data entries per page.
totalElements	Int	Total number of data entries.
totalPages	Int	Total number of pages calculated based on (pageSize).