BlackBerry Dynamics SDK for iOS

Development Guide

11.2

Contents

What is the BlackBerry Dynamics SDK?

The BlackBerry Dynamics SDK provides a powerful set of tools that you can use to create secure productivity apps for a BlackBerry UEMdomain. The SDK leverages the full capabilities of the secure BlackBerry Dynamics platform, so you can focus on building your apps rather than learning how to secure, deploy, and manage those apps.

The BlackBerry Dynamics SDK is available for all major development platforms. It allows you to leverage many valuable services, including secure communication, securing data in file systems and databases, inter-app data exchange, presence, push, directory lookup, single sign-on authentication, identity and access management, and more.

This guide will provide:

- Information about supported features
- Development requirements and prerequisites
- Instructions for installing, configuring, and using the SDK
- Considerations for key platform features
- Information about the sample apps provided with the SDK
- Testing and troubleshooting guidance
- Guidance for deploying your app

This guide is intended for intermediate and experienced developers with an understanding of how to create apps for the intended platform. It is not a basic tutorial.

BlackBerry Dynamics API reference

The BlackBerry Dynamics SDK API reference describes the available interfaces, classes, methods, and much more. The API reference for each SDK platform is available at https://developers.blackberry.com/us/en/resources/api-reference.html.

Key features of the BlackBerry Dynamics SDK

This section provides more information about key features of the BlackBerry Dynamics SDK. It does not detail the complete feature set. For more information about the full list of supported features and APIs, see the BlackBerry Dynamics SDK API reference for your platform.

For more information about the requirements and prerequisites to support platform-specific features, see Requirements and support for platform-specific features.

The implementation of some of the features discussed in this section will depend on how the UEM administrator has configured your organization's servers, network, and other infrastructure components. Contact the administrator to clarify whether there are components of a feature that are configured or managed using the management server.

Activation

Infrastructure and enterprise activation

After a BlackBerry Dynamics app is installed on a user's device, the user must activate the app in order to use it. The activation process registers the app with the management server and gives the app access to the full capabilities of the BlackBerry Dynamics platform. The activation process ensures that all end users are fully authorized and permitted to use the app.

Users can activate a BlackBerry Dynamics app manually using an activation password, QR code, or access key provided by the administrator or obtained from UEM Self-Service, by using the UEM Client, through a third-party IDP, such as Active Directory or Okta, or by using the Easy Activation feature described below.

For more information about activating BlackBerry Dynamics apps, see the Activation section in the GDAndroid class reference or GDiOS class reference and Managing BlackBerry Dynamics apps in the UEM Administration content.

Easy Activation

Easy Activation simplifies the process of activating multiple BlackBerry Dynamics apps on a user's device. With Easy Activation, a device user only needs to activate the first BlackBerry Dynamics app on their device; when the user installs additional BlackBerry Dynamics apps, the user can choose to delegate the activation process to the previously activated app. Any BlackBerry Dynamics app can be an activation delegate, but priority is given to the app that is configured as the authentication delegate.

Easy Activation is automatically enabled for all BlackBerry Dynamics apps that are produced by BlackBerry. To enable Easy Activation for your custom BlackBerry Dynamics app, the UEM administrator must specify the app package ID (Android) or bundle ID (iOS) in the BlackBerry Dynamics app settings in the management console. Contact your organization's administrator to provide this information. For instructions for specifying the package ID or bundle ID for an app, see Manage settings for a BlackBerry Dynamics app in the UEM Administration content.

On the application side, Easy Activation is enabled by default by the BlackBerry Dynamics Runtime.

For more information, see the Easy Activation section in the BlackBerry Dynamics Security White Paper.

iOS user enrollment and DEP activation enhancements

The BlackBerry Dynamics SDK for iOS version 8.0 and later and UEM version 12.13 and later feature the following activation enhancements for iOS user enrollment and DEP:

- MDM enrollment and the activation of BlackBerry Dynamics apps doesn't require the UEM Client.
- After a new device is enrolled on UEM, UEM prompts the user to install the BlackBerry Dynamics app that is configured as the authentication delegate (that app must be assigned to the user). When the user opens this app for the first time, it activates automatically. The user can then activate additional BlackBerry Dynamics apps using Easy Activation.

Programmatic activation

The programmatic activation feature enables a BlackBerry Dynamics app to activate without any user interaction and without displaying activation prompts or progress screens. This can be useful when targeting your apps to a consumer audience or for developing apps for devices that have limited or no means of user input.

For more information about programmatic activation, see `programmaticActivityInit` in the BlackBerry Dynamics SDK for Android API Reference or `programmaticAuthorize` in the BlackBerry Dynamics SDK for iOS API Reference.

Note the following implementation details:

- Decide whether you want users to specify a password to unlock a BlackBerry Dynamics app after the initial activation. You can use programmatic activation while still requiring users to type a password to unlock the app. This setting is configured in a BlackBerry Dynamics profile in UEM.
- To activate the app, your application server must use the BlackBerry Web Services REST APIs to retrieve the user credentials and to generate an access key. You may need to create a new UEM user account or to lookup an existing user account. See the User resource in the BlackBerry Web Services REST API reference for the available REST APIs that can be used to create a user, lookup a user, and to generate an access key.
- Pass the user credentials to the app and call `programmaticActivityInit` or `programmaticAuthorize` with the retrieved credentials, setting `ShowUserInterface` to false.

- For Android, receive the broadcast event `GD_STATE_ACTIVATION_ACTION` to track activation progress from `NotActivated` to `InProgress` to `Activated`. You can choose to display a progress indicator during this short period.
- For iOS, observe `GDState.BBActivationState` to track activation progress from `NotActivated` to `InProgress` to `Activated`. You can choose to display a progress indicator during this short period.
- Once activation completes, the user is prompted to set a password (unless you've configured the BlackBerry Dynamics profile to not require a password). The app should wait for the `GD_STATE_AUTHORIZED_ACTION` notification.
- You can use `configureUI` (Android/iOS) to customize the UI of the password screen (for example, with a custom logo and colors).

Secure storage

Secure file system

BlackBerry Dynamics apps store data in a secure, encrypted file system. For more information, see BlackBerry Dynamics File I/O Package in the BlackBerry Dynamics SDK for Android API reference, or GDFileManager and GDFileHandle in the BlackBerry Dynamics SDK for iOS API reference.

Core data

BlackBerry Dynamics apps can store Core Data objects in a secure, encrypted store. For more information, see GDPersistentStoreCoordinator in the API reference.

Secure SQL database

BlackBerry Dynamics apps can leverage a secure SQL database that stores and encrypts data on the user's device. The secure SQL database is based on the SQLite library.

For more information, see the BlackBerry Dynamics SQL Database page in the BlackBerry Dynamics SDK API reference (Android/iOS).

Secure communication

The BlackBerry Dynamics platform enables secure data exchange between a BlackBerry Dynamics app on an end user's device and a back-end application server on the Internet or behind the enterprise firewall. Any communication through the enterprise firewall uses the secure BlackBerry Dynamics proxy infrastructure. One app can communicate with multiple application servers.

To learn more about the programming interfaces for secure communication, see GDSocket and GDHttpClient in the BlackBerry Dynamics SDK for Android API reference, or GDSocket, GDURLLoadingSystem, and NSURLSession Support in the BlackBerry Dynamics SDK for iOS API reference.

AppKinetics

AppKinetics, or Inter-Container Communication (ICC), is a method for securely exchanging data and commands between two BlackBerry Dynamics apps on the same device. The exchange uses a consumer-provider model: one app initiates a service request that the other app receives and responds to as a service provider.

For more information about AppKinetics, see the Inter-Container Communication Package in the BlackBerry Dynamics SDK for Android API reference, or GDService and GDServiceClient in the BlackBerry Dynamics SDK for iOS API reference.

Shared Services Framework

BlackBerry Dynamics apps can communicate with each other and application servers using the Shared Services Framework, a collaboration system that is defined by two components: one that provides a service and another that consumes the service.

The provider can be a client-side service, which is a BlackBerry Dynamics app that uses the GDService APIs (Android/iOS), or a server-side service that is provided by an application server or other remote system. The service is consumed by a BlackBerry Dynamics app that communicates with the provider using AppKinetics (a proprietary BlackBerry ICC protocol) for client-side services or a protocol such as HTTPS for server-side services.

The typical steps that are required to consume a service:

1. Service discovery: The BlackBerry Dynamics app (the consumer) queries for service providers using the GDAndroid.getServiceProvidersFor API or the GDiOS.getServiceProvidersFor API. Service discovery is optional but recommended for both types of services because it respects user entitlements and permissions.
2. Provider selection: The consuming app selects the provider. This is handled by the app code.
3. Service request: The consuming app sends a service request to the provider using the GDServiceClient API (Android/iOS) for client-side services or TCP sockets or HTTP over BlackBerry Dynamics secure communication (Android/iOS) for server-side services.
4. Service response: The consuming app receives the provider response using the same interface that was used for the request (the GDServiceClient API (Android/iOS) for client-side services or BlackBerry Dynamics secure communication (Android/iOS) for server-side services).

Client-side services can be used offline and are ideal if the service requires specific user interaction.

Server-side services can be provided by a clustered application server and are ideal if the server software already exists outside of the BlackBerry Dynamics platform.

Client-side and server-side services both require user entitlement in the management console.

If you want your custom BlackBerry Dynamics app to use the Shared Services Framework, the UEM administrator must specify the app package ID (Android) or bundle ID (iOS) in the BlackBerry Dynamics app settings in the management console. Contact your organization's administrator to provide this information. For instructions for specifying the package ID or bundle ID for an app, see Manage settings for a BlackBerry Dynamics app in the *UEM Administration Guide*.

Sample apps that are included with the SDK demonstrate how to use the Shared Services Framework. For more information about how to use the Shared Services Framework, see the following resources:

- GDAndroid.getServiceProvidersFor
- GDiOS.getServiceProvidersFor
- GDService (Android/iOS)
- GDServiceClient (Android/iOS)
- BlackBerry Dynamics Service Definition (Android/iOS)
- Definitions and descriptions of published services

Server-side services can use the Push Channel API (Android/iOS) to send notifications to BlackBerry Dynamics apps. The channel is end-to-end secure at the same level as BlackBerry Dynamics secure communication. As a result, the BlackBerry Dynamics app does not need to poll the application server, which decreases the load on both the app and the application server. Any application server that is a service provider can use the Push Channel.

Data Leakage Prevention

The BlackBerry UEM administrator can use Data Leakage Prevention (DLP) settings in BlackBerry Dynamics profiles (UEM) to configure data protection standards, including enabling or disabling copy and paste between BlackBerry Dynamics apps and non-BlackBerry Dynamics apps, screen captures, dictation, FIPS, and more.

Contact your organization's administrator to configure DLP standards as necessary for your custom BlackBerry Dynamics apps.

Note the following for the different platforms of the SDK:

SDK platform	Notes
BlackBerry Dynamics SDK for Android	The SDK provides the following classes to manage the secure copy, cut, and paste of data: ClipboardManager, GDTextView, GDEditText, GDAutoCompleteTextView, and GDSearchView. The secure copy-cut-paste sample app demonstrates uses of the secure clipboard. Due to current DLP controls, the setOnReceiveContentListener method is not supported in the following components and widgets: • com.good.gd.widget.GDAutoCompleteTextView • com.good.gd.widget.GDEditText • com.good.gd.widget.GDMultiAutoCompleteTextView • com.good.gd.widget.GDSearchView • com.good.gd.widget.GDTextView • com.good.gd.widget.GDWebView • com.blackberry.bbwebview.BBWebView
BlackBerry Dynamics SDK for iOS	The BlackBerry Dynamics Runtime can secure or block text in transit to or from the clipboard, depending on the configuration of the DLP settings. The Runtime secures text by encrypting it when it is cut or copied to the clipboard and decrypting it when it is pasted. These operations are handled automatically by the Runtime, so no development changes are required.
BlackBerry Dynamics SDK for Cordova	You do not need to make any development changes to support DLP for Cordova apps for iOS or Android. Note that for Android Cordova apps, secure cut, copy, and paste is currently blocked.

User authentication

BlackBerry UEM offers the following options to adjust the user experience for accessing BlackBerry Dynamics apps.

Fingerprint and biometric authentication

Various forms of biometric authentication are supported by the BlackBerry Dynamics SDK, including fingerprint authentication and for Android and Touch ID and Face ID for iOS. The BlackBerry UEM administrator can use a BlackBerry Dynamics profile (UEM) to enable biometric authentication. Contact your organization's administrator to enable and configure these features.

For more information, see BlackBerry Dynamics and Fingerprint Authentication.

Authentication delegation

The BlackBerry UEM administrator can configure up to three BlackBerry Dynamics apps on users' devices to act as an authentication delegate (a primary, secondary, and tertiary delegate). When a user opens any BlackBerry Dynamics app, the device will display the login screen of the authentication delegate app. After the user logs in successfully, all of the BlackBerry Dynamics apps on the device are unlocked. The user does not need to enter a password again until the idle timeout is reached.

If you want your custom BlackBerry Dynamics app to be an authentication delegate, the UEM administrator must specify the app package ID (Android) or bundle ID (iOS) in the BlackBerry Dynamics app settings in the management console. Contact your organization's administrator to provide this information. For instructions for specifying the package ID or bundle ID for an app, see Manage settings for a BlackBerry Dynamics app in the *UEM Administration Guide*.

The administrator configures one or more authentication delegate using a BlackBerry Dynamics profile. It is a best practice to configure the most commonly used app as the authentication delegate. Contact your organization's administrator to configure one or more authentication delegates.

Note: If the administrator configures a secondary authentication delegate, the administrator must notify users that if they delete the primary authentication delegate app, the user must unlock the secondary delegate app and set the app password again so that it can be used to authenticate any additional BlackBerry Dynamics apps. The same requirement applies if a tertiary delegate is configured and the primary and secondary delegate apps are deleted.

Do not require a password

Enabled using a BlackBerry Dynamics profile, this setting removes the password login for BlackBerry Dynamics apps. Users cannot choose whether to use a password.

Do not enable authentication delegation and this setting in the same profile or policy set. This feature is supported in UEM 12.7 or later. If the setting is enabled and then disabled at a later date, users are prompted to create a password the next time they log in to a BlackBerry Dynamics app.

You can use the GDAndroid.getInstance().canAuthorizeAutonomously() or [GDiOS sharedInstance].canAuthorizeAutonomously method to check if this feature is enabled. See the GDInteraction sample app (Android) or the SecureStore sample app (iOS) for examples of this method.

Bypass the app unlock screen

Enabled in the UEM Client settings for a specific BlackBerry Dynamics app (UEM), this setting allows an app to completely bypass the password login screen.

For more information and programming guidance, see the Bypass Unlock Developer Guide.

Background Authorize for iOS

Background Authorize is a restricted API that allows a recently locked BlackBerry Dynamics app to use the principal BlackBerry Dynamics APIs (such as secure storage and secure communication) when the app is running in the background.

This feature can be useful in scenarios where the app has stopped unexpectedly and is started in the background in response to an APNS message (for example, a new email). If Background Authorize is enabled, the app can download new data and store it in the secure container. When the user brings the app to the foreground they can authorize and immediately access the data (for example, messages).

To access this restricted API, submit a request to the BlackBerry Dynamics Registrar program at BlackBerryDynamicsRegistrar@blackberry.com.

For more information about this feature, see the Background Authorize Developer Guide.

Background Authorize for Android

GDAndroid.canAuthorizeAutonomously allows BlackBerry Dynamics apps to background unlock, receive state callback, and use credential-protected storage. The app can use canAuthorizeAutonomously() to check if it is possible to use background unlock, and if possible, authorize with serviceInit().

Web Authentication

The SDK supports ASWebAuthenticationSession. The BlackBerry Dynamics implementation of ASWebAuthenticationSession utilizes BlackBerry Dynamics secure communication and secure storage for cookies. To protect enterprise credentials from being stored in the iOS keychain, the device user will not be able to use the Safari saved passwords feature in the embedded webview.

Initialize an instance of ASWebAuthenticationSession in your app to allow user authentication through a web service, including those operated by a third party. The page will open in a secure, embedded webview in iOS, or the users default browser (if it supports web authentication sessions) on macOS. For more information, see Authenticating a User Through a Web Service.

Administrative controls

The BlackBerry UEM administrator can use various server settings, policies, and profiles to manage BlackBerry Dynamics apps and ensure that app usage meets the organization's security standards. Consult with your organization's administrator to ensure that your custom apps adhere to the configured settings in the management console.

For more information, see Controlling BlackBerry Dynamics on users devices, Enforcing compliance rules for devices, and Managing BlackBerry Dynamics apps in the *UEM Administration Guide*.

You also have the option to add new management policies and settings that are specific to your custom BlackBerry Dynamics app to the UEM management console so that they can be configured and applied to users by UEM administrators. For more information, see Adding custom policies for your app to the UEM management console.

Advanced security features with CylancePROTECT Mobile

The BlackBerry Dynamics SDK integrates the CylancePROTECT library to support CylancePROTECT Mobile for UEM. CylancePROTECT is a licensed service that offers a suite of features that enhances UEM's ability to detect, prevent, and resolve security threats without disrupting the productivity of your workforce. CylancePROTECT is configured and managed by the UEM administrator. No additional development or integration effort is required if your organization wants to leverage the CylancePROTECT features for custom BlackBerry Dynamics apps.

CylancePROTECT uses a combination of advanced technologies, including:

- The cloud-based CylanceINFINITY service that uses sophisticated AI and machine learning to identify malware and unsafe URL
- The UEM server that provides a complete device management and compliance infrastructure for your organization
- BlackBerry apps that monitor and enforce security standards at the device and user level

The seamless integration of these technologies establishes a secure ecosystem where data is protected and malicious activities are identified at all endpoints and eliminated proactively.

CylancePROTECT includes the following features:

- Malware detection for Android apps (including BlackBerry Dynamics apps) that are uploaded to UEM for internal deployment
- Malware detection on Android devices
- Sideloaded app detection on iOS and Android devices
- Safe browsing with BlackBerry Dynamics apps
- Insecure network detection on iOS and Android devices.
- Insecure Wi-Fi access point detection on Android devices.
- Integrity checking for BlackBerry Dynamics apps on iOS devices using the Apple DeviceCheck framework
- Hardware certificate attestation for BlackBerry Dynamics apps on Android devices

For more information about CylancePROTECT, see the BlackBerry Protect documentation.

Data collection and metrics with BlackBerry Analytics

BlackBerry Analytics is a cloud-based portal that you can use to view information about the BlackBerry Dynamics apps and devices that are used in your organization's environment. Previously, the BlackBerry Analytics functionality was offered in a separate SDK that you could integrate with your BlackBerry Dynamics apps. In SDK version 8.0 and later, BlackBerry Analytics functionality is now included in the BlackBerry Dynamics SDK.

For more information about BlackBerry Analytics, see the BlackBerry Analytics documentation. For more information about integrating BlackBerry Analytics with your BlackBerry Dynamics apps, see Integrating BlackBerry Analytics.

Requirements and support for platform-specific features

This section provides the software requirements for using the SDK, as well as prerequisites that are required to support platform-specific features.

Software requirements

iOS development

Item	Requirement
Compatibility with previous versions of the SDK	The latest release of the BlackBerry Dynamics SDK for iOS is compatible with these previous releases of the SDK: • 9.2.x • 9.1.x • 10.0.x • 10.1.x • 10.2.x • 11.0.x
Deployment target	iOS 15 or later
Xcode	Xcode 13 or 14
Support for Mac Silicon M1 devices	• BlackBerry Dynamics apps using SDK version 11.0 or later are supported for Mac Silicon devices. • BlackBerry Dynamics apps running on Mac Silicon devices are reported as Mac apps in the UEM management console. • UEM administrators can control access to BlackBerry Dynamics apps using the macOS compliance settings available in the management console. • To support Easy Activation and authentication delegation, you will need to provide the macOS bundle ID for the app in the Apps settings in myAccount.
Supported programming languages	• Objective-C • Swift 4, 4.2, 5
Supported Internet Protocols	• IPv4 • IPv6

Item	Requirement
Info.plist requirements	In the Info.plist file, add the key "Privacy - Camera Usage Description" with the value "Allow camera usage to scan a QR code". This is not required if the app already uses the camera for its own purposes. For ISV apps, it is recommended that you add the following usage descriptions (or your own custom descriptions) to the Info.plist file to inform the user why the app requires location permission: • NSLocationWhenInUseUsageDescription: "Allow BlackBerry to collect location data, including Wi-Fi address and IP address, and usage patterns only when the app is in use. You may change this setting at any time from your device settings." • NSLocationAlwaysUsageDescription: "Always allow BlackBerry to collect location data, including Wi-Fi address and IP address, and usage patterns, even when the app is not in use. You may change this setting at any time from your device settings." • NSLocationAlwaysAndWhenInUseUsageDescription: "Allow BlackBerry to collect location data, including Wi-Fi address and IP address, and usage patterns both when the app is and is not in use. You may change this setting at any time from your device settings."
Native bundle ID	If you develop a BlackBerry Dynamics app for use on both iPhone and iPad devices, use a single native bundle ID for all variations of the app. UEM will only accept a single native bundle ID.
Keychain group sharing for multiple apps	Keychain group sharing allows groups of apps to share information that is stored on a device's keychain. Keychain group sharing is required when you are developing multiple inter-related apps. The setting is part of a project's build. To enable keychain group sharing in an Xcode project, open the project file, navigate to the app target Capabilities tab, and turn on Keychain Sharing. You may be asked for your developer password and to choose a development team. The provisioning profiles for each app must come from the same team and must share the same App ID prefix (see row below). For the Keychain Group, specify `com.good.gd.data`. Also, if you intend to use crypto tokens in your app, specify `com.apple.token`. If the settings for keychain group sharing change, it is recommended to do a fresh reinstall of the new version of the app instead of upgrading the old version. This ensures that the new keychain settings take effect.
App ID prefix	An App ID prefix is a unique ID that groups a collection of apps and enables those apps to share keychain and UIPasteboard data. Apps that share keychain data must have a common App ID prefix from Apple. For more information, see Technical Note TN2311: Managing Multiple App ID Prefixes. The Apple App ID prefix is completely independent of the BlackBerry Dynamics entitlement ID.

Item	Requirement
BlackBerry Dynamics Launcher Library	The BlackBerry Dynamics Launcher is a user-friendly interface that allows users to easily access and switch between BlackBerry Dynamics apps, configure app settings, and take advantage of other useful features. For more information, see the BlackBerry Dynamics Launcher Framework documentation. The BlackBerry Dynamics SDK and the BlackBerry Dynamics Launcher Library are mutually dependent. See the BlackBerry Dynamics SDK for iOS Release Notes for the required version of the BlackBerry Dynamics Launcher Library.
Restricted key prefix	The key prefix "blackberry" is reserved by BlackBerry and should not be used for key values, key attributes, or key elements. For more information and examples, see the Application Policies Definition in the appendix of the API Reference.

Using an entitlement ID and version to uniquely identify a BlackBerry Dynamics app

BlackBerry Dynamics apps are uniquely identified by a BlackBerry Dynamics entitlement ID (GDApplicationID) and entitlement version (GDApplicationVersion). The entitlement ID and entitlement version are used to manage end-user entitlement in the management console. The values are also used for app publishing and service provider registration.

These values are specified in the assets/settings.json file for Android or in the Info.plist file for iOS.

You must define both the entitlement ID and the entitlement version for all of your BlackBerry Dynamics apps, regardless of whether you use the Shared Services Framework. The same entitlement ID must be used to represent the app across all platforms.

For more information about setting and checking the entitlement ID and version, the proper format to use for these values, and other requirements and considerations:

- See the GDAndroid Class reference in the BlackBerry Dynamics SDK for Android API Reference, especially these sections:
  - Identification
  - Indirect Authorization Initiation
  - void authorize (GDAppEventListener eventListener) throws GDInitializationError
- See the GDiOS Class reference in the BlackBerry Dynamics SDK for iOS API Reference, especially these sections:
  - Identification
  - Authorization
  - - (void) authorize: (id< GDiOSDelegate > _Nonnull) delegate

Relationship between the entitlement ID and version and native identifiers

The entitlement ID (GDApplicationID) and entitlement version (GDApplicationVersion) of a BlackBerry Dynamics app are different from the native identifiers that are required by the app OS platform. The native identifiers for Android are the packageName and packageVersion values in the AndroidManifest.xml file. The native identifiers

for iOS are the CFBundleIdentifier and CFBundleVersion in the Info.plist file. The values of the entitlement ID and entitlement version and the platform native identifiers can be the same, but do not have to be.

To take advantage of BlackBerry Dynamics features such as Easy Activation, authentication delegation, certificate sharing, the Shared Services Framework, and more, the UEM administrator must specify the entitlement ID and version and the native identifier (package name or bundle ID) for a custom BlackBerry Dynamics app in the management console. For more information, see Add an internal BlackBerry Dynamics app entitlement and Manage settings for a BlackBerry Dynamics app in the *UEM Administration Guide*.

The native identifiers for your custom BlackBerry Dynamics app should be unique, especially with respect to apps that are available through public app stores. Duplicate native identifiers can prevent the proper installation or upgrade of an app.

You must change the native app version if you want to distribute a new version of the app. You only need to change the entitlement version if the app starts to provide a new shared service or shared service version, or if the app stops providing a shared service or shared service version.

FIPS compliance

It is a best practice to make your BlackBerry Dynamics apps compliant with U.S. Federal Information Processing Standards (FIPS) 140-2.The BlackBerry Dynamics SDK distribution contains FIPS canisters and tools.

The BlackBerry UEM administrator enables FIPS compliance using a BlackBerry Dynamics profile (UEM). If enabled, BlackBerry Dynamics apps must start in FIPS-compliant mode. The SDK determines whether a service is running in FIPS mode when the app communicates with the server to receive policies.

FIPS compliance enforces the following constraints:

- The use of MD4 and MD5 are prohibited. As a result, access to NTLM-protected or NTLM2-protected web pages and files is blocked.
- In secure socket key exchanges with ephemeral keys, with servers that are not configured to use Diffie-Hellman keys of sufficient length, BlackBerry Dynamics retries with static RSA cipher suites.

Note:

- When you enable FIPS compliance, user certificates must use encryption that meets FIPS standards. If a user tries to import a certificate with encryption that is not compliant, the user receives an error message indicating that the certificate is not allowed and cannot be imported.
- For iOS, when you build for testing with the x86 64-bit simulator, FIPS mode is not enforced. As a result, you might see a difference in behavior with the simulator compared to actual operation. BlackBerry recommends that you always test your app on actual iOS hardware and not rely exclusively on the simulation.
- If you use the SDK dynamic framework, FIPS linking is not required.

Declaring a URL type to support BlackBerry Dynamics features

A BlackBerry Dynamics app for iOS devices must declare a URL type so that it can be discovered by other apps on the same device. This enables AppKinetics, which is required for many BlackBerry Dynamics features. The URL type and schemes are declared in the app's Info.plist file.

The URL type must be the same as the app's native bundle ID. Within the URL type declaration, the following URL schemes must be declared. For example, if the native bundle ID of the app is `com.example.gd.myapp` and its entitlement version (GDApplicationVersion) is 1.0.0.0, then the declared URL type is `com.example.gd.myapp` and the schema declarations are as follows:

Format	Description	Example
`com.good.gd.discovery.enterprise`	Always required for enterprise apps (not required for ISV apps)	Exactly as shown
`<bundle_ID>.sc3`	Always required	`com.example.gd.myapp.sc3`
`<bundle_ID>.sc2`	Enables an app to use authentication delegation and is required for all BlackBerry Dynamics apps	`com.example.gd.myapp.sc2`
`<bundle_ID>.sc2.<GDApplicationVersion>`	Required only if your app provides a discoverable service	`com.example.gd.myapp.sc2.1.0.0.0`

App UI restrictions

The BlackBerry Dynamics Runtime monitors the app UI to enforce the configuration of enterprise profiles or policies from BlackBerry UEM. For example, a BlackBerry Dynamics profile may require the user to enter a password when the app transitions from the background to the foreground, or it may lock the app UI after a certain period of inactivity.

For a complete explanation of the restrictions and requirements that the app UI must follow, see Application User Interface Restrictions in the API Reference.

Requirements and prerequisites for iOS platform features

This section provides specific requirements or considerations to support features of the iOS platform in your BlackBerry Dynamics apps.

Support for Touch ID

Touch ID is a fingerprint recognition system for some iOS devices.

Touch ID can be allowed for user authentication in BlackBerry Dynamics apps in addition to standard password authentication. For more information about Touch ID, see BlackBerry Dynamics and Fingerprint Authentication in the BlackBerry Developers for Enterprise Apps portal.

Support for Face ID

The BlackBerry Dynamics SDK for iOS version 4.0 and later supports Face ID. An administrator can enable or disable the feature in a BlackBerry Dynamics profile.

Each app must add the NSFaceIDUsageDescription key to the Info.plist file. For more details about Face ID, see the Build-Time Configuration appendix in the API Reference.

Support for WKWebView

The BlackBerry Dynamics SDK for iOS version 4.2 and later supports secure WKWebView for displaying interactive web content.

Note the following support details:

- The SDK supports multiple WKWebView instances. The instances must be created programmatically.
- The SDK supports loading WKWebView from UIStoryBoard. To avoid any possible data leaks, you must load UIStoryBoard with the WKWebView component after the SDK is initialized.
- The supported versions of iOS require JavaScript injection by the BlackBerry Dynamics Runtime.
- The secure Fetch API is supported.
- Synchronous XMLHttpRequests are supported for iOS 12.2 and later, but the GET method is supported for iOS 13.1 and later only.
- The SDK supports the use of the cache to search for valid cached data for resources loaded by WKWebView.

The SDK's implementation of secure WKWebView currently supports:

- Loading HTTP and HTTPS data
- Redirection
- Basic, Digest, NTLM, Kerberos, and ClientCertificate authentication
- Cookies
- Video and audio playback
- Asynchronous XHR requests
- HTML5 non-persistent local storage
- HTML5 persistent local storage
- Sending the following types of body data using XMLHttpRequest: ArrayBuffer, Blob, FormData, URLSearchParams, USVString
- WebRTC
- HTTP/2

Note:

WebSockets are supported in WKWebView with the following limitations:

- ServerTrust challenge
- bufferedAmount property for WebSocket JavaScript object
- Basic, Digest, NTLM, and Kerberos authentication
- Extensions

The SDK's implementation of secure WKWebView does not currently support:

- The JavaScript sendBeacon API
- The following Data Leakage Prevention (DLP) settings from BlackBerry UEM for long-press or 3D touch actions:
  - Do not allow copying data from BlackBerry Dynamics apps into non BlackBerry Dynamics apps
  - Do not allow copying data from non BlackBerry Dynamics apps into BlackBerry Dynamics apps
- HTML attributes for a link tag (for example, preconnect)

WKWebView: Unsupported methods and properties

Class: WKContentWorld

WKContentWorld is new in iOS 14 and is not yet supported by the SDK. The SDK supports only WKContentWorld.pageWorld:.

Class: WKDownload

WKDownload is new in iOS 14.5 and is not yet supported by the SDK. If a developer attempts to use WKDownload via a 3rd party framework, the console will produce the following error messages:

- `WKGD: Attempt to use WKDownload interface detected! Dynamics SDK does not support WKDownload.`
- `WKGD: Attempt to start download via WKDownload interface detected! Dynamics SDK does not support WKDownload.`
- `WKGD: Attempt to resume download via WKDownload interface detected! Dynamics SDK does not support WKDownload.`

Class: WKDownloadDelegate

WKDownloadDelegate is new in iOS 14.5 and is not yet supported by the SDK.

Class: WKFindConfiguration

WKFindConfiguration is new in iOS 14 and is not yet supported by the SDK.

Class: WKFindResult

WKFindResult is new in iOS 14 and is not yet supported by the SDK.

Class: WKNavigationDelegate

Method	Details
webView(_:authenticationChallenge:shouldAllowDeprecatedTLS:)	This method is new in iOS 14 and is not yet supported by the SDK.

Class: WKPDFConfiguration

WKPDFConfiguration is new in iOS 14 and is not yet supported by the SDK.

Class: WKUserContentController

Method	Details
addScriptMessageHandler:contentWorld:name:	This method is new in iOS 14. The SDK supports only WKContentWorld.pageWorld.
addScriptMessageHandlerWithReply: contentWorld:name:	This method is new in iOS 14. The SDK supports only WKContentWorld.pageWorld.

Method	Details
removeAllScriptMessageHandlersFromContentWorld:	New in iOS 14. You cannot use this method with WKContentWorld.pageWorld. If you need to remove the script message handler from WKContentWorld.pageWorld, remove the handler by name using removeScriptMessageHandlerForName:contentWorld:.
removeAllScriptMessageHandlers	New in iOS 14. You should remove the handler by name using removeScriptMessageHandlerForName:contentWorld:.

Class: WKWebView

Method or property	Details
evaluateJavaScript:inFrame:inContentWorld: completionHandler:	This method is new in iOS 14. The SDK supports only WKContentWorld.pageWorld.
callAsyncJavaScript:arguments:inFrame:inContentWorld: completionHandler:	This method is new in iOS 14. The SDK supports only WKContentWorld.pageWorld.
createPDFWithConfiguration:completionHandler:	After calling this function, the user gets NSData in the completion handler. To store data in the secure container, use GDFileManager. To securely send data through the network, use NSURLSession.
createWebArchiveDataWithCompletionHandler:	After calling this function, the user gets NSData in the completion handler. To store data in the secure container, use GDFileManager. To securely send data through the network, use NSURLSession.
findString:withConfiguration:completionHandler:	This method is new in iOS 14 and is not yet supported by the SDK.
loadFileURL(_:allowingReadAccessTo:)	Files can be loaded from a bundle or from the App folder, but they can't be loaded from an encrypted, secured container that is protected by the SDK.
loadFileRequest:allowingReadAccessToURL:	Files can be loaded from a bundle or from the App folder, but can't be loaded from an encrypted, secured container that is protected by the SDK.
mediaType	This property is new in iOS 14 and is not yet supported by the SDK.

Method or property	Details
printOperationWithPrintInfo:	This method is for macOS only and is not currently supported by the SDK.
resumeDownloadFromResumeData:completionHandler:	This method is new in iOS 14.5 and is not supported by the SDK. The SDK generates a warning when it is called.
serverTrust	The SDK returns nil for this property.
startDownloadUsingRequest:completionHandler:	This method is new in iOS 14.5 and is not supported by the SDK. The SDK generates a warning when it is called.
uiDelegate	The SDK returns its own internal delegate. If you set a custom delegate the property will work as expected.

Class: WKWebViewConfiguration

Method or property	Details
limitsNavigationsToAppBoundDomains	This property is new in iOS 14 and is not currently supported by the SDK.

Class: WKNavigationAction

Method or property	Details
shouldPerformDownload	This property is new in iOS 14.5 and is not supported by the SDK.

Class: WKNavigationDelegate

Enumeration Case	Details
WKNavigationActionPolicyDownload	This enumeration case is new in iOS 14.5 and is not supported by the SDK.
WKNavigationResponsePolicyDownload	This enumeration case is new in iOS 14.5 and is not supported by the SDK.

Class: WKUIDelegate

Method	Details
webView:requestDeviceOrientationAndMotionPermissionForOrigin: initiatedByFrame:decisionHandler:	This method is not supported by the SDK.

Class: WKUserContentController

Method	Details
removeAllUserScripts()	The SDK injects its own scripts, so calling this method will break how the SDK supports WKWebView.

Class: NSAttributedString (NSAttributedStirngWebKitAdditions)

Method	Details
loadFromHTMLWithFileURL:options:completionHandler:	Files can be loaded from a bundle or from the App folder, but can't be loaded from an encrypted, secured container that is protected by the SDK.

Class: WKPreferences

Method	Details
fraudulentWebsiteWarningEnabled	The SDK always sets this value to NO and the user can't change this value. The SDK provides an alternative mechanism to validate URLs. For more information, see Safe browsing with BlackBerry Dynamics apps.

Support for SFSafariViewController

The BlackBerry Dynamics SDK for iOS supports secure SFSafariViewController for displaying web interfaces within BlackBerry apps.

Note the following support details:

- The use of SFSafariViewController to load a preview of a web page is secured with BlackBerry Dynamics secure communication. You must add SafariServices.framework to your Xcode project.
- The SDK does not support authentication using SFSafariViewController.
- Navigation control is disabled when using SFSafariViewController with the SDK. The SDK supports a single view only.
- The SDK does not support signing in with SFSafariViewController to Microsoft Azure Active Directory Connect (using MSAL) is not supported.

Support for the Apple Universal Clipboard

The SDK supports secure cut, copy, and paste operations using the Apple Universal Clipboard. Users can copy and move data between BlackBerry Dynamics apps and devices that follow Apple Continuity system requirements. The apps and devices must be activated for the same user account on the same instance of BlackBerry UEM.

For more information about the Apple Universal Clipboard, see support.apple.com to read Set up Universal Clipboard.

Unsupported iOS features

The following features are not supported by the BlackBerry Dynamics SDK:

Feature	Details
BitCode	BitCode is an intermediate, architecture-independent binary object format that allows developers to submit a "machine neutral" app to Apple for a final, architecture-dependent build. The intermediate nature of BitCode is not compatible with the cryptographic requirements of the BlackBerry Dynamics SDK, which relies on the delivery of libraries or other modules that are cryptographically signed at build-time. The signatures of the libraries and modules are validated at runtime to ensure integrity. If you enable BitCode and try to build an app, a build error will indicate that the SDK does not support BitCode.
App Extensions	The BlackBerry Dynamics SDK does not support app extensions such as Sirikit.

Supported TLS protocols and cipher suites

The SDK uses CURL 7.70.0, supporting the TLS protocols and cipher suites of TLS version 1.3.

Steps to get started with the BlackBerry Dynamics SDK

Step	Action
1	Use the BlackBerry Dynamics SDK framework.
2	Add the BlackBerry Dynamics SDK event-handler skeleton manually or by using notifications.
3	Consult the BlackBerry Dynamics SDK API reference for instructions for implementing the desired features of the BlackBerry Dynamics platform. See the sample apps included in the SDK package for examples of how to implement key features. For additional guidance and code examples, see the Getting started workflow for BlackBerry developers.
4	Review the optional features that you can integrate.
5	Test and debug your app. The SDK allows you to test in enterprise simulation mode and supports automated testing.
6	Review configuration recommendations and then deploy your app.
7	Optionally, deploy certificates to the BlackBerry Dynamics apps on users' devices.

Using the BlackBerry Dynamics SDK framework

The BlackBerry Dynamics SDK is currently offered as a dynamic framework, consisting of the BlackBerryDynamics.xcframework and two BlackBerryCerticom xcframeworks. Follow the tasks below to configure your existing apps and new apps to use the BlackBerry Dynamics SDK.

Prepare an existing BlackBerry Dynamics app to use the dynamic framework

Complete the steps below if you have an existing BlackBerry Dynamics app that uses the static BlackBerry Dynamics SDK library (GD.Framework).

1. Delete the *~/Library/Application Support/BlackBerry* folder.
2. In the Xcode Project Navigator panel for your project, delete all of the references to **GDAssets.bundle** and **GD.framework** or **GD.xcframework**.
3. Update the project's import paths to reference **BlackBerryDynamics** instead of **GD**.
   - For example, for C-style import or includes, change **GD/** to **BlackBerryDynamics/GD/**.
   - For module-based imports, replace **GD** with **BlackBerryDynamics**.

Note: GD_C_NETUtility and sqlite3 have been moved from GD to GD_C modules.

4. In the **default.xcconfig** file, remove the 3 lines starting with FIPS_PACKAGE, LDPLUSPLUS, and LD.
5. In the **Frameworks, Libraries and Embedded Content** list of the project target, add the following items:
   - BlackBerryDynamics.xcframework
   - BlackBerryCerticom.xcframework
   - BlackBerryCerticomSBGSE.xcframework

   Set all of the above frameworks to **Embed and Sign** so that Xcode will change the project file by adding the frameworks to the Embed Frameworks step in the Build Phases and will configure the Runpath Search Paths accordingly.
6. In the **Build Settings** of the project target, update **Framework Search Paths** to include the directory of the BlackBerry framework files.
7. In the **Header Search Paths**, include the following entry: <directory_with_framework_files>/BlackBerryDynamics.xcframework/Headers

After you finish: Configure a new or existing BlackBerry Dynamics app to use the dynamic framework.

Configure a new or existing BlackBerry Dynamics app to use the dynamic framework

Before you begin:

- If required, Prepare an existing BlackBerry Dynamics app to use the dynamic framework.

Choose one of the following methods to configure the app to use the BlackBerry Dynamics SDK dynamic framework. BlackBerry recommends using the CocoaPods method.

Method	Steps

Use CocoaPods	**a.** If you don't have an existing pod file, use the following commands to create one:
	`cd '<project_directory>'`
	`pod init`
	b. Add the following reference to the BlackBerry Dynamics SDK pod:
	`pod 'BlackBerryDynamics'`
	c. To add the BlackBerry Dynamics ATSL to the target app for UI tests, add the following pod:
	`pod 'BlackBerryDynamicsAutomatedTestSupportLibrary'`
	For more information, see Add automated testing to your BlackBerry Dynamics iOS app.
	d. If default.xcconfig is used by your application, add the following to the end of your Podfile:
	`post_install do |installer|`
	`  # Append configuration from default.xcconfig to configuration generated by Cocoapods`
	`  if Dir.glob("**/default.xcconfig").first()`
	`  system("find Pods -name 'Pods-*.xcconfig' -exec sh -c 'cat `find . -name default.xcconfig` >> $1' -- {} \;")`
	`  end`
	`end`
	e. Install the pods, then open your .xcworkspace file to see the project in Xcode:
	`$ pod install`
	`$ open <your-project>.xcworkspace`

Use the files available in the SDK package	**a.** Extract the contents of the **BlackBerry_Dynamics_SDK_for_iOS_<*version*>_dylib.tar.gz** file from the SDK package to a directory. **b.** In the **Frameworks, Libraries and Embedded Content** list of the project target, add **BlackBerryDynamics.xcframework, BlackBerryCerticom.xcframework, BlackBerryCerticomSBGSE.xcframework**. Set all of the above frameworks to **Embed and Sign** so that Xcode will change the project file by adding the frameworks to the Embed Frameworks step in the Build Phases and will configure the Runpath Search Paths accordingly. **c.** In the **Build Settings** of the project target, update **Framework Search Paths** to include the directory of the BlackBerry framework files. **d.** In the **Header Search Paths**, include the following entry: <*directory_with_framework_files*>/BlackBerryDynamics.xcframework/Headers

Add the event-handler skeleton manually

The application-delegate object `UIApplicationDelegate` manages the iOS app life cycle using events. You can decide how to implement the life cycle of events; the steps below provide an example that is based loosely on the Apple Breadcrumbs sample app (with differences in declarations and other key areas). Several of the sample apps included in the SDK also demonstrate the event handling lifecycle.

1. Specify the entitlement ID (`GDApplicationID`) and entitlement version (`GDApplicationVersion`) in the **Info.plist** file. See Using an entitlement ID and version to uniquely identify a BlackBerry Dynamics app.
2. Import `GDiOS.h` and implement a skeleton `GDiOSDelegate` protocol. The following examples for Objective-C and Swift rely on a boolean variable named `started`. An alternative approach is to declare the variable in your project's **.m** file.

   Objective-C:

   ```
   // AppDelegate.h
   #import <UIKit/UIKit.h>
   #import <GD/GDiOS.h>

   @interface AppDelegate : UIResponder <UIApplicationDelegate, GDiOSDelegate> {
       BOOL started;
   }

   @property (strong, nonatomic) UIWindow *window;
   @property (strong, nonatomic) GDiOS *good;

   - (void)onAuthorized:(GDAppEvent *)anEvent;
   - (void)onNotAuthorized:(GDAppEvent *)anEvent;

   @end
   ```

   Swift:

   ```
   class AppDelegate : UIResponder , UIApplicationDelegate,  GDiOSDelegate {

       var window: UIWindow?
       var good: GDiOS?
       var started: Bool = false
   }
   ```

3. Use `GDAppEvent` to process events. Move the application launch code from `didFinishLaunchingWithOptions` to the `GDAppEvent` handler method.

   Objective-C:

   ```
   // AppDelegate.m
   - (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:
   (NSDictionary *)launchOptions {
       // Override point for customization after application launch.
       self.good = [GDiOS sharedInstance];
       _good.delegate = self;
       started = NO;
       //Show the BlackBerry Authentication UI.
       [_good authorize];

       return YES;
   }
   ```

```
#pragma mark BlackBerry Dynamics Delegate Method
- (void)handleEvent:(GDAppEvent *)anEvent
{
    /* Called from _good when events occur, such as system startup. */
    switch (anEvent.type)
    {
        case GDAppEventAuthorized:
        {
            [self onAuthorized:anEvent];
            break;
        }
        case GDAppEventNotAuthorized:
        {
            [self onNotAuthorized:anEvent];
            break;
        }
        case GDAppEventRemoteSettingsUpdate:
        {
            // handle app config changes
            break;
        }
        case GDAppEventPolicyUpdate:
        {
            // handle app policy changes
            break;
        }
        case GDAppEventServicesUpdate:
        {
            // handle services changes
            break;
        }
        case GDAppEventEntitlementsUpdate:
        {
            // handle entitlements changes
            break ;
        }
        default:
            NSLog(@"Unhandled Event");
            break;
    }
}
```

Swift:

```
func application(_ application: UIApplication, didFinishLaunchingWithOptions
 launchOptions: [UIApplication.LaunchOptionsKey: Any]?) -> Bool {

    //Call BlackBerry Dynamics to authorise the app
    self.good = GDiOS.sharedInstance()

    self.good!.delegate = self
    self.started = false

    // Show the Good Authentication UI.
    self.good!.authorize()

    return true
}

func handle(_ anEvent: GDAppEvent) {
```

```
    switch anEvent.type {
        case .authorized:
            onAuthorized(anEvent: anEvent)
            break

        case .notAuthorized:
            self.onNotAuthorized(anEvent: anEvent)
            break

        case .remoteSettingsUpdate:
            if started {

                let storyBoard: UIStoryboard = UIStoryboard(name: "Main",
 bundle: nil)
                let newViewController =
 storyBoard.instantiateViewController(withIdentifier: "ShareViewController")
 as! ShareViewController
                self.window?.rootViewController = newViewController
                newViewController.refreshUI()


            }
            break

        case .servicesUpdate:
            //A change to services-related configuration.
            break

        case .policyUpdate:
            if started {
                NotificationCenter.default.post(name:
 NSNotification.Name(rawValue: "AppPolicyUpdated"), object: nil);
            }
            break


        default:
            print("handleEvent \(anEvent.message)")
        }
}
```

4. Special-case the `onNotAuthorized` events. Verify that the app can handle authorization errors or functionality like remote wipe, a lockout, or blocking events. Implement these events in the `onNotAuthorized` function.

   Objective-C:

```
- (void)onNotAuthorized:(GDAppEvent *)anEvent
{
    /* Handle the BlackBerry Libraries not authorized event. */
    switch (anEvent.code) {
        case GDErrorActivationFailed:
        case GDErrorProvisioningFailed:
        case GDErrorPushConnectionTimeout:
        case GDErrorSecurityError:
        case GDErrorAppDenied:
        case GDErrorBlocked:
        case GDErrorWiped:
        case GDErrorRemoteLockout:
        case GDErrorPasswordChangeRequired: {
```

```
            // A condition has occurred denying authorization, an application
 may wish to log these events
            NSLog(@"onNotAuthorized %@", anEvent.message);
            break;
        }
        case GDErrorIdleLockout: {
            // idle lockout is benign & informational
            break;
        }
        default:
            NSAssert(false, @"Unhandled not authorized event");
            break;
    }
}
```

Swift:

```
func application(_ application: UIApplication, didFinishLaunchingWithOptions
 launchOptions: [UIApplication.LaunchOptionsKey: Any]?) -> Bool {

func onNotAuthorized(anEvent:GDAppEvent ) {
    /* Handle the Good Libraries not authorized event. */
    switch anEvent.code {
        case .errorActivationFailed: break
        case .errorProvisioningFailed: break
        case .errorPushConnectionTimeout: break
        case .errorSecurityError: break
        case .errorAppDenied: break
        case .errorAppVersionNotEntitled: break
        case .errorBlocked: break
        case .errorWiped: break
        case .errorRemoteLockout: break
        case .errorPasswordChangeRequired:
            // an condition has occured denying authorization, an application
 may wish to log these events
            print("onNotAuthorized \(anEvent.message)")
        case .errorIdleLockout:
            // idle lockout is benign & informational
            break

        default: assert(false, "Unhandled not authorized event");
    }
}
```

5. On authorization, start the app. Initialize and start the UI with the `onAuthorized` function. The `GDErrorNone` event is returned by the BlackBerry Dynamics Runtime whenever a container is opened and no error occurs.

   Objective-C:

   ```
   - (void)onAuthorized:(GDAppEvent *)anEvent
   {
       /* Handle the BlackBerry Libraries authorized event. */
       switch (anEvent.code) {
           case GDErrorNone: {
               // started was declared in first step.
               if (!started) {
                   // launch application UI here
                   started = YES;
               }
               break;
   ```

```
        } default:
            NSAssert(false, @"Authorized startup with an error");
            break;
    }
}
```

Swift:

```
func onAuthorized(anEvent:GDAppEvent ) {
    /* Handle the Good Libraries authorized event. */
    switch anEvent.code {
        case .errorNone:
            if !self.started {
                //Show the User UI
                self.started = true
                UIApplication.shared.windows.filter
 {$0.isKeyWindow}.first?.rootViewController = UIStoryboard(name: "Main",
 bundle: nil).instantiateViewController(withIdentifier: "tabBarController")
            }
        default:
            assert(false, "Authorized startup with an error")
            break
        }
}
```

Add the event-handler skeleton using notifications

The application-delegate object `UIApplicationDelegate` manages the iOS app life cycle using events. You can decide how to implement the life cycle of events; the steps below provide an example that is based loosely on the Apple Breadcrumbs sample app (with differences in declarations and other key areas). Several of the sample apps included in the SDK also demonstrate the event handling lifecycle.

1. Specify the entitlement ID (`GDApplicationID`) and entitlement version (`GDApplicationVersion`) in the **Info.plist** file. See Using an entitlement ID and version to uniquely identify a BlackBerry Dynamics app.
2. In the info.plist file of the app, add a dictionary entry with the name `BlackBerryDynamics`. For the value, add another dictionary entry containing a single entry with the key `CheckEventReceiver` and its Boolean value set to `NO`.

```
<plist version="1.0">
    <dict>
    ...
        <key>BlackBerryDynamics</key>
        <dict>
            <key>CheckEventReceiver</key>
            <false/>
        </dict>
    ...
    </dict>
</plist>
```

3. Import `GDiOS.h` and `GDState.h`. This example relies on a boolean variable named `'hasAuthorized'`. An alternative approach is to declare the variable in your project's **.m** file.

   Objective-C:

```
// AppDelegate.h
```

```
#import <UIKit/UIKit.h>
#import <GD/GDiOS.h>
#import <GD/GDState.h>

@interface AppDelegate : UIResponder <UIApplicationDelegate> {
    BOOL hasAuthorized;
}

@property (strong, nonatomic) UIWindow *window;

- (void)handleStateChange:(GDState *)state;
- (void)onAuthorized;
- (void)onNotAuthorized:(GDAppResultCode)code;

@end
```

Swift:

```
class AppDelegate : UIResponder , UIApplicationDelegate,  GDiOSDelegate {
    var hasAuthorized: Bool = false
}
```

4. Make your `AppDelegate` an observer for state changing notifications.

Objective-C:

```
// AppDelegate.m

- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:
(NSDictionary *)launchOptions {
    // Override point for customization after application launch.
    [self addStateObserverUsingNotificationCenter];
    hasAuthorized = NO;
    [[GDiOS sharedInstance] authorize];

    return YES;
}

#pragma mark - BlackBerry Dynamics observer for state changes (Notification
 Center)

- (void)addStateObserverUsingNotificationCenter
{
    // register to receive notification center notifications for GD state
 changes
    [[NSNotificationCenter defaultCenter] addObserver:self

 selector:@selector(receiveStateChangeNotification:)
 name:GDStateChangeNotification object:nil];
}

- (void) receiveStateChangeNotification:(NSNotification *) notification
{
    if ([[notification name] isEqualToString:GDStateChangeNotification]) {
        NSDictionary* userInfo = [notification userInfo];
        NSString *propertyName = [userInfo
 objectForKey:GDStateChangeKeyProperty];
        GDState* state = [userInfo objectForKey:GDStateChangeKeyCopy];
```

```
        // For the purposes of this example, we want to log a different message
 so that it is known
        // these calls are coming from Notification Center
        if ([propertyName isEqualToString:GDKeyIsAuthorized]) {
            NSLog(@"receiveStateChangeNotification - isAuthorized: %@",
 state.isAuthorized ? @"true" : @"false");
            [self handleStateChange:state];

        } else if ([propertyName isEqualToString:GDKeyReasonNotAuthorized]) {
            NSLog(@"receiveStateChangeNotification - reasonNotAuthorized: %ld",
 (long) state.reasonNotAuthorized);

        } else if ([propertyName isEqualToString:GDKeyUserInterfaceState]) {
            NSLog(@"receiveStateChangeNotification - userInterfaceState: %ld",
 (long) state.userInterfaceState);

        } else if ([propertyName isEqualToString:GDKeyCurrentScreen]) {
            NSLog(@"receiveStateChangeNotification - currentScreen: %ld",
 (long) state.currentScreen);
        }
    }
}
```

Swift:

```
func application(_ application: UIApplication, didFinishLaunchingWithOptions
 launchOptions: [UIApplication.LaunchOptionsKey: Any]?) -> Bool {
    //Register for notifications from Blackberry Dynamics
    NotificationCenter.default.addObserver(self, selector:
 #selector(receiveStateChangeNotification(_:)), name:
 Notification.Name.GDStateChange, object: nil)

    hasAuthorized = false

    //Call BlackBerry Dynamics to authorise the app
    GDiOS.sharedInstance().authorize()

    return true
}

@objc func receiveStateChangeNotification(_ notification: NSNotification) {
    if (notification.name == Notification.Name.GDStateChange) {
        let userInfo = notification.userInfo
        let propertyName = userInfo?[GDStateChangeKeyProperty] as! String
        let state = userInfo?[GDStateChangeKeyCopy] as! GDState

        // For the purposes of this example, we want to log a different message
 so that it is known
        // these calls are coming from Notification Center
        if (propertyName == GDKeyIsAuthorized) {
            print("receiveStateChangeNotification - isAuthorized:
 \(state.isAuthorized)")
            handleStateChange(state)
        } else if (propertyName == GDKeyReasonNotAuthorized) {
            print("receiveStateChangeNotification - reasonNotAuthorized:
 \(state.reasonNotAuthorized)")
        } else if (propertyName == GDKeyUserInterfaceState) {
            print("receiveStateChangeNotification - userInterfaceState:
 \(state.userInterfaceState)")
        } else if (propertyName == GDKeyCurrentScreen) {
```

```
            print("receiveStateChangeNotification - currentScreen:
 \(state.currentScreen)")
        }
    }
}
```

5. Use `GDState` to handle the state of the BlackBerry Dynamics Runtime.

   Objective-C:

```
// AppDelegate.m

#pragma mark - Good Dynamics handler methods for state changes

- (void)handleStateChange:(GDState *)state
{
    if (state.isAuthorized) {
        NSLog(@"gdState: authorized");
        [self onAuthorized];

    } else {
        GDAppResultCode errorCode = [state reasonNotAuthorized];
        NSLog(@"gdState: not authorized, error:%ld", (long) errorCode);
        [self onNotAuthorized:errorCode];

    }
}

- (void)onAuthorized
{
    if (!hasAuthorized) {
        // launch application UI here
        hasAuthorized = YES;

    }
}

- (void)onNotAuthorized:(GDAppResultCode)code
{
    /* Handle the Good Libraries not authorized event. */

    switch (code)
    {
        case GDErrorActivationFailed:
        case GDErrorProvisioningFailed:
        case GDErrorPushConnectionTimeout:
        case GDErrorSecurityError:
        case GDErrorAppDenied:
        case GDErrorAppVersionNotEntitled:
        case GDErrorBlocked:
        case GDErrorWiped:
        case GDErrorRemoteLockout:
        case GDErrorPasswordChangeRequired:
        {
            // an condition has occured denying authorization, an application
 may wish to log these events
            NSLog(@"gdState: not authorized, error:%ld", (long) code);
            break;
        }
        case GDErrorIdleLockout:
        {
```

```
            // idle lockout is benign & informational
            break;
        }
        default:
            NSAssert(false, @"Unhandled not authorized event");
            break;
    }
}
```

Swift:

```
func handleStateChange(_ state: GDState) {
    if state.isAuthorized {
        print("gdState: authorized");
        onAuthorized()
    } else {
        print("gdState: not authorized, error \(state.reasonNotAuthorized)");
        onNotAuthorized(errorCode: state.reasonNotAuthorized)
    }
}


func onAuthorized() {
    // Handle the Good Libraries authorized event
    if (!hasAuthorized) {
        // launch application UI here
        hasAuthorized = true

    }
}


func onNotAuthorized(errorCode: GDAppResultCode) {
    // Handle the Good Libraries not authorized event
    switch errorCode {
    case .errorActivationFailed: break
    case .errorProvisioningFailed: break
    case .errorPushConnectionTimeout: break
    case .errorSecurityError: break
    case .errorAppDenied: break
    case .errorAppVersionNotEntitled: break
    case .errorBlocked: break
    case .errorWiped: break
    case .errorRemoteLockout: break
    case .errorPasswordChangeRequired:
        // an condition has occured denying authorization, an application may
 wish to log these events
        print("gdState: not authorized, error \(errorCode)")
    case .errorIdleLockout:
        // idle lockout is benign & informational
        break
    default: assert(false, "Unhandled not authorized event");
    }
}
```

Integrating optional features

This section provides more information about integrating optional features into your custom BlackBerry Dynamics apps.

Preventing password autofill in the app UI

Password autofill is an iOS feature that automatically provides suggestions for a user's password or other privacy data in an app. This feature does not satisfy the security standards of the BlackBerry Dynamics platform and can lead to the exposure of sensitive data.

The BlackBerry Dynamics SDK disables the password autofill feature for all screens in BlackBerry Dynamics apps, but it does not block the feature in the app UI. BlackBerry provides an open source sample on GitHub that demonstrates how to prevent the password autofill feature in the UI. For more information, see AutoFill blocking solution for Password AutoFill in UITextField.

This method is independent of and does not rely on the BlackBerry Dynamics SDK. Note that this solution applies only to UITextField.

Enforcing local compliance actions

The BlackBerry Dynamics SDK includes the following APIs that you can use to block or unblock a user's access to the UI of a BlackBerry Dynamics app locally:

- GDiOS.executeBlock
- GDiOS.executeUnblock

You can use these APIs to temporarily prevent access to an app under certain conditions. For example, if the user accesses a public Wi-Fi network that is not trusted, you can use GDiOS.executeBlock to prevent access to the app until the user is on a trusted Wi-Fi network. While the app UI is blocked, the app's network activity and container storage access is not affected.

You can use GDiOS.executeBlock to display a message to the user that explains why access to the app has been blocked and how the user can restore compliance and unblock the UI.

The RemoteDB sample app has been updated to demonstrate the use of these APIs.

Note: It is possible to circumvent a UI block if the user is able to restore a backup that was created before the block occurred. Take this condition into account when you develop and test your app.

Adding custom policies for your app to the UEM management console

You can add new management policies and settings that are specific to your custom BlackBerry Dynamics app to the BlackBerry UEM management console so that they can be configured and applied to users by UEM administrators. The new management policies that you define are in addition to the full suite of BlackBerry Dynamics and device management policies already offered by UEM.

You can add a custom policy for a BlackBerry Dynamics app by creating and uploading an application policy definition file to the management console. For implementation details and more information about the file format and elements, see the Application Policies Definition appendix in the API Reference (Android/iOS) and the BlackBerry Dynamics App Policies Technical Brief.

For instructions for uploading the file to the management console, see Manage settings for a BlackBerry Dynamics app in the *UEM Administration Guide*.

Add a watermark to the screens in a BlackBerry Dynamics app

To protect against data leakage, you can implement an app policy that adds a watermark to the screens in a BlackBerry Dynamics app. The watermark includes the user's username and the current date and time. This watermark deters users from taking photos of potentially sensitive information, while also giving your organization a means to trace the source of a data leak.

To implement this feature, you must use BlackBerry Dynamics SDK version 8.0 or later, and you must implement a BlackBerry Dynamics application policy in UEM to enable or disable the feature.

Note: The watermark does not apply to views that appear outside of the interactive use of an application (for example, notifications and widgets). Review the data leakage protection settings available in the BlackBerry Dynamics profile to configure the necessary protections for sensitive information.

1. Create a BlackBerry Dynamics app policy definition file for your app. For implementation details and more information about the file format and elements, see the Application Policies Definition appendix in the API Reference (Android/iOS) and the BlackBerry Dynamics App Policies Technical Brief.
2. Add a setting definition for the DLP watermark policy. The name of the key must be **blackberry.security.EnableDLPWatermark** and the default value should be **false**.

   Example:

   ```
   <setting name="blackberry.security.EnableDLPWatermark">
       <checkbox>
           <key>blackberry.security.EnableDLPWatermark</key>
           <label>Enable DLP Watermark</label>
           <value>false</value>
       </checkbox>
   </setting>
   ```

3. Include the DLP watermark policy setting in the structural section of the file. The pview elements must not include a key attribute.

   Example:

   ```
   <pview>
       <pview type="tabbed">
           <title>Data Leakage Prevention Watermark</title>
           <desc>Add a faint background watermark across all application screens
    with users 'username' and the current date/time.</desc>
           <pe ref="blackberry.security.EnableDLPWatermark" />
       </pview>
   </pview>
   ```

4. Coordinate with the UEM administrator to upload the app policy for your app in UEM. See Manage settings for a BlackBerry Dynamics apps (App configuration > Upload a template).

The UEM administrator can now enable the DLP watermark feature.

Allow unencrypted data to be copied to the pasteboard

The BlackBerry Dynamics platform offers comprehensive data leakage prevention controls, but in certain cases you may want to allow an app to copy unencrypted data to the pasteboard so that it can be consumed by another non-BlackBerry Dynamics app.

To implement this feature, you must use BlackBerry Dynamics SDK version 8.0 or later, and you must implement a BlackBerry Dynamics application policy in UEM that allows the administrator to enable or disable the feature. See the GDNativePasteboardAccess class reference for details about the client-side implementation.

1. Create a BlackBerry Dynamics app policy definition file for your app. For implementation details and more information about the file format and elements, see the Application Policies Definition appendix in the API Reference and the BlackBerry Dynamics App Policies Technical Brief.
2. Add a setting definition for the Native Pasteboard Access policy. The name of the key must be **blackberry.security.EnableNativePasteboardAcess** and the default value should be **false**.

   Example:

   ```
   <setting name="blackberry.security.EnableNativePasteboardAccess">
       <checkbox>
           <key>blackberry.security.EnableNativePasteboardAccess</key>
           <label>Enable iOS Native Pasteboard Access</label>
           <value>false</value>
       </checkbox>
   </setting>
   ```

3. Include the Native Pasteboard Access policy setting in the structural section of the file.

   Example:

   ```
   <pview>
       <pview type="tabbed" key="iOSNativePasteboard">
           <title>iOS Native Pasteboard</title>
           <desc>Permit this application to use the GDNativePasteboardAccess
    API. This function allows pasteboard items to be written to and read from the
    pasteboard in the clear while data leakage prevention is active.</desc>
           <pe ref="blackberry.security.EnableNativePasteboardAccess" />
       </pview>
   </pview>
   ```

4. Coordinate with the UEM administrator to upload the app policy for your app in UEM. See Manage settings for a BlackBerry Dynamics apps (App configuration > Upload a template).

The UEM administrator can now enable the feature.

Replace the default splash screen for inactive apps

When a BlackBerry Dynamics app becomes inactive, a BlackBerry branded splash screen is displayed by default. You can use either of the following methods to replace the default splash screen with a custom splash screen:

- **Replace the splash screen at build-time:** In your app's plist file, add a value using the key UILaunchStoryboardName. The BlackBerry Dynamics Runtime will try to instantiate a splash screen from the storyboard with this name. By default, the name in Xcode will be LaunchScreen.
- **Replace the splash screen at runtime:** Customize the splash screen during the life-cycle of the app (for example, to handle the user experience for specific system events) by using the class GDSplashScreenCustomizer.h and its associated delegate. This class allows you to replace the splash

screen with a custom view controller. Follow the instructions below to create an app specific policy so the UEM administrator can enable or disable the custom splash screen.

1. Create a BlackBerry Dynamics app policy definition file for your app. For implementation details and more information about the file format and elements, see the Application Policies Definition appendix in the API Reference and the BlackBerry Dynamics App Policies Technical Brief.
2. Add a setting definition for the custom splash screen policy. Specify a name for the key, for example **BD_SDK_AllowCustomSplashScreen**. The default value should be **false**.

   Example:

   ```
   <setting name="BD_SDK_AllowCustomSplashScreen">
       <checkbox>
           <key>BD_SDK_AllowCustomSplashScreen</key>
           <label>Allow app to display custom splash screens replacing the
    default</label>
           <value>false</value>
       </checkbox>
   </setting>
   ```

3. Include the custom splash screen policy setting in the structural section of the file.

   Example:

   ```
   <pview type="tabbed">
       <title>BlackBerry Dynamics Features</title>
       <pview>
           <heading>Custom Splash Screens</heading>
           <desc>Describe the custom splash screen feature and what data is
    displayed on the custom splash screens.</desc>
           <pe ref="BD_SDK_AllowCustomSplashScreen" />
       </pview>
   </pview>
   ```

4. Coordinate with the UEM administrator to upload the app policy for your app in UEM. See Manage settings for a BlackBerry Dynamics apps (App configuration > Upload a template). The administrator can now enable the custom splash screen feature.

Prompt the user to update a BlackBerry Dynamics app

Users should always upgrade to the latest available version of a BlackBerry Dynamics app to benefit from the latest security protections, features, and fixes. If a user has turned off automatic app updates on their device, the user runs the risk of remaining on an older version of an app that does not have the latest enhancements and protections. You can implement an app policy that will enable your BlackBerry Dynamics apps, on startup, to detect whether the user should upgrade to the latest available version. The BlackBerry Dynamics Runtime provides a notification using the ThreatStatus interface only if the user is running an older version of the app.

To implement this feature, you must use BlackBerry Dynamics SDK version 8.0 or later, and you must implement a BlackBerry Dynamics application policy in UEM. Note that the UEM administrator cannot view or change the policy configuration. The minimum required version and the app download URL are configured using hidden fields in the application policy.

The ThreatApplicationSecurity (Android/iOS) class in the ThreatStatus API (Android/iOS) provides a warning. The isAppVersionOutOfDate() method (Android/iOS) indicates if the current version of the app that is installed on the device is lower than the minimum required version. A string with the URL to download the latest version is returned using getAppUpdateURL() (Android/iOS).

1. Create a BlackBerry Dynamics app policy definition file for your app. For implementation details and more information about the file format and elements, see the Application Policies Definition appendix in the API Reference and the BlackBerry Dynamics App Policies Technical Brief.
2. Add the following settings:
   - **blackberry.security.appInfo.android.appMinVersion**: This is the minimum required versionCode from the Android manifest file.
   - **blackberry.security.appInfo.iOS.appMinVersion**: This is the minimum required bundle version as defined in the Xcode info.plist file.
   - **blackberry.security.appInfo.android.appUpdateURL**: This is the URL that the user can use to download your Android app.
   - **blackberry.security.appInfo.iOS.appUpdateURL**: This is the URL that the user can use to download your iOS app.

   Example:

```
<setting name="android_appMinVersion">
    <hidden>
        <key>blackberry.security.appInfo.android.appMinVersion</key>
        <value>16</value>
    </hidden>
</setting>
<setting name="iOS_appMinVersion">
    <hidden>
        <key>blackberry.security.appInfo.iOS.appMinVersion</key>
        <value>20</value>
    </hidden>
</setting>
<setting name="iOS_appUpdateURL">
    <hidden>
        <key>blackberry.security.appInfo.iOS.appUpdateURL</key>
        <value>https://apps.apple.com/us/app/blackberry-work/id890656632</
value>
    </hidden>
</setting>
<setting name="android_appUpdateURL">
    <hidden>
        <key>blackberry.security.appInfo.android.appUpdateURL</key>
        <value>https://play.google.com/store/apps/details?id=com.good.gcs</
value>
    </hidden>
</setting>
```

3. Include the settings in the structural section of the file.

   Example:

```
<pview>
    <!-- App Minimum Version Security Configuration -->
    <pe ref="android_appMinVersion"/>
    <pe ref="android_appUpdateURL"/>
    <pe ref="iOS_appMinVersion"/>
    <pe ref="iOS_appUpdateURL"/>
</pview>
```

4. Coordinate with the UEM administrator to upload the app policy for your app in UEM. See Manage settings for a BlackBerry Dynamics apps (App configuration > Upload a template).

Adding a custom logo and colors with the branding API

You can use the branding API to add a custom logo and colors to the app UI. For more information, see (void) configureUIWithLogo in the API Reference.

Using zero sign-on for SaaS services through BlackBerry Enterprise Identity

Your custom BlackBerry Dynamics apps can leverage zero sign-on (ZSO) for SaaS through BlackBerry Enterprise Identity. For more information, see the BlackBerry Enterprise Identity docs.

Integrating BlackBerry Enterprise Mobility Server services

This section covers the general approach for programming with the BlackBerry Dynamics SDK and the BlackBerry Enterprise Mobility Server services. The approach consists of two parts:

- Programming an app to interact with the desired BEMS services
- Entitling users to the necessary applications

BEMS services conform to the Shared Services Framework. A service consists of two applications: A program that provides the service, and an app that consumes the service. BEMS is the service provider that must be configured for use in BlackBerry UEM. You create the app that consumes this service.

BEMS services APIs

The BEMS services are described in the BEMS API Reference Guides.

Programming your service consumer app

You must define a unique BlackBerry Dynamics app ID for your application (for complete details, see Using an entitlement ID and version to uniquely identify a BlackBerry Dynamics app). The BlackBerry Dynamics SDK has functions to discover services, and each BEMS service has specific programming interfaces.

To discover the BEMS services, use `GDServiceType`. This API and other APIs for shared services are described in other sections of this guide and in the BlackBerry Dynamics API reference.

After your consumer app discovers the service, the way the app communicates with the service depends on the service definition.

Note: Most BEMS services run over SSL (HTTPS) on port 8443. Be sure your consumer application connects to the correct server and port.

Discovering the BlackBerry Enterprise Mobility Server services

Described here is a general approach to using the BlackBerry Dynamics SDK and Server-based Services Framework to programmatically discover the Docs services offered by your BEMS installation.

Item	Description
Service identifier	First you need to know the service identifier and version. For more information about the available services, see Mobile Services.
Service discovery	Next, code a service discovery query in your application program. See the `getServiceProvidersFor` API in the `GDAndroid`, `GDiOS`, and `GDMac` classes.
Server cluster	The result of the service discovery query is an array of GDServiceProvider objects. Each object corresponds to a BlackBerry Dynamics entitlement ID that is registered as a provider of the service. Your best result is that the array has one element. If the array is empty, it means that the current end user isn't entitled to any App ID that provides the service. In that case, your app shouldn't use the service. If the array has more than one element, it means that the end user is entitled to more than one GD App ID that provides the service (likely a configuration error by the enterprise). Your app would have to pick one of the GD App IDs, or try all of them, or prompt the user to select. In the `GDServiceProvider` object, there is a `serverCluster` attribute. It contains an array of `GDAppServer` objects, each of which tells you the address and port number of a server, and the priority of that instance within the cluster.
Server selection	If the `serverCluster` array has only one element, then server selection is trivial. Use the server address and port number of the first element. If the `serverCluster` array is empty, that indicates an enterprise configuration error. If the `serverCluster` array has more than one element, then you must implement a server selection algorithm. A sample algorithm is given on the `GDAndroid`, `GDiOS`, and `GDMac` pages in the BlackBerry Dynamics API reference, in the `getApplicationConfig` section. The algorithm is the same for the BlackBerry Dynamics SDK for Android and for the BlackBerry Dynamics SDK for iOS. The recommended selection algorithm is as follows. For each priority value in the list, starting with the highest: • Select a server that has that priority, at random. • Attempt to connect to the server. • If the connection succeeds, use that server. • If the connection fails, try another server at the same priority, at random. • If there are no more untried servers at that priority, try the servers at the next lower priority.

Enabling microphone and camera support with WebRTC

As of BlackBerry Dynamics SDK version 10.2, access to WebRTC is disabled by default. As a result, the following WebRTC API's are blocked:

- getUserMedia
- enumerateDevices

- getSupportedConstraints
- getDisplayMedia
- RTCPeerConnection
- RTCSessionDescription
- RTCIceCandidate
- RTCPeerConnectionIceEvent
- RTCPeerConnectionIceErrorEvent
- RTCCertificate
- RTCRtpSender
- RTCRtpReceiver
- RTCRtpTransceiver
- RTCDtlsTransport
- RTCIceTransport
- RTCTrackEvent
- RTCSctpTranspor
- RTCDataChannel
- RTCDataChannelEvent
- RTCDTMFSender
- RTCDTMFToneChangeEvent
- RTCStatsReport
- RTCErrorEvent

You can use a setting in the app configuration to turn WebRTC on and off. By enabling WebRTC, you can provide microphone and camera support that is required for certain services (for example, Zoom or WebEx). When you enable WebRTC, users will be prompted to approve the use of the camera and microphone when they are using a supported service.

If you want to enable WebRTC in your app, you must select **Allow WebRTC** in the app configuration settings for your app on the UEM server. WebRTC traffic is routed directly to the internet instead of through the BlackBerry Dynamics secure tunnel regardless of the network connectivity profile settings because they do not apply to WebRTC traffic. Because WebRTC traffic is not secured by the SDK, there are potential security risks that can occur for users that approve the use of WebRTC. If you require the use of WebRTC and can accept this risk, you can implement an app policy to enable or disable WebRTC usage.

Note: WebRTC policy changes cannot be applied during runtime. Any policy changes will be applied after the app is restarted or the WKWebView instance is reinitialized.

To add a WebRTC policy for your app, do the following:

1. Create a BlackBerry Dynamics app policy definition file for your app. For implementation details and more information about the file format and elements, see the Application Policies Definition appendix in the API Reference and the BlackBerry Dynamics App Policies Technical Brief.
   (Optional) Provide an example. If you include a screen shot, include alt text in the alt element.
2. Add a setting definition for the WebRTC. The name of the key must be **k_allow_web_rtc**, and the default value should be **false**. For example:

```
<setting name="blackberry.security.EnableWebRTC">
    <checkbox>
        <key>features.k_allow_web_rtc</key>
        <label>Enable WebRTC</label>
        <value>false</value>
    </checkbox>
</setting>
```

3. Include the WebRTC policy setting in the structural section of the file. The pview elements must not include a key attribute. For example:

```
<pview>
    <pview type="tabbed">
        <title>WebRTC (Real-Time Communication)</title>
        <desc>To improve security and privacy WebRTC is disabled in BlackBerry
 Dynamics. Optionally enable WebRTC connections to enable audio and video peer-
peer communication from this application.</desc>
        <pe ref="blackberry.security.EnableWebRTC" />
    </pview>
</pview>
```

4. Coordinate with the UEM administrator to upload the app policy for your app in UEM. For more information, see Manage settings for a BlackBerry Dynamics apps (App configuration > Upload a template).

The UEM administrator can now enable the WebRTC feature.

Integrating BlackBerry Analytics

BlackBerry Analytics is a cloud-based portal that you can use to view information about the BlackBerry Dynamics apps and devices that are used in your organization's environment. Previously, the BlackBerry Analytics functionality was offered in a separate SDK that you could integrate with your BlackBerry Dynamics apps. In SDK version 8.0 and later, BlackBerry Analytics functionality is now included in the BlackBerry Dynamics SDK.

BlackBerry Analytics functionality is automatically enabled when you use the BlackBerry Dynamics SDK version 8.0 or later. To enable data collection and use of the BlackBerry Analytics portal, your organization must purchase the BlackBerry Analytics entitlement, and the UEM administrator must assign the entitlement to users or groups. For more information about the BlackBerry Analytics requirements, entitlement, and accessing the portal, see the BlackBerry Analytics documentation. If you are upgrading an app to use SDK version 8.0 and it currently uses the separate BlackBerry Analytics SDK, see Remove the previous integration with the separate BlackBerry Analytics SDK.

For more information about the BlackBerry Analytics APIs that are integrated in the BlackBerry Dynamics SDK, see the BlackBerryAnalytics class (Android/iOS) in the API reference.

Remove the previous integration with the separate BlackBerry Analytics SDK

If you previously integrated the separate BlackBerry Analytics SDK with your BlackBerry Dynamics app, when you upgrade to use SDK version 8.0, do the following to remove the previous integration:

1. Remove **BAFBlackberryAnalytics.framework** from the Link Binary With Libraries build phase.
2. Remove **BAFBlackberryAnalyticsAssets.bundle** from the Copy Bundle Resources build phase.
3. Remove the following method calls within your project:
   - `BlackberryAnalytics.setStage()`
   - `BlackberryAnalytics.hostInitializationCompleted()`
4. Change the import for the `BAFBlackberryAnalytics.h` header from `#import <BAFBlackberryAnalytics/BAFBlackberryAnalytics.h>` to `#import <GD/BAFBlackberryAnalytics.h>`.

Once you upgrade to SDK version 8.0, no additional development effort is required to enable your BlackBerry Dynamics app to use BlackBerry Analytics. To enable data collection and use of the BlackBerry Analytics portal, your organization must purchase the BlackBerry Analytics entitlement and the UEM administrator must assign the entitlement to users or groups. For more information about the BlackBerry Analytics requirements, entitlement, and accessing the portal, see the BlackBerry Analytics documentation.

Using the BlackBerry Analytics REST API

You can use the BlackBerry Analytics REST API to retrieve metrics and event data from BlackBerry Analytics. You can use this data in a number of ways. For example, you can create your own custom reports.

To use the BlackBerry Analytics REST API, you should be familiar with REST APIs that rely on JSON bodies for payload. You must also have a registered account that you can use to access BlackBerry Analytics at https://analytics.blackberry.com. For instructions for setting up BlackBerry Analytics and for creating an account, see the BlackBerry Analytics Administration Guide.

The BlackBerry Analytics REST API reference is available on the Analytics Portal. The API documentation includes examples of various requests.

Get a JWT token for REST API calls

1. Get a refresh token from the BlackBerry myAccount online service. Use the same myAccount credentials that you use to log in to the BlackBerry Analytics portal. Refresh tokens are valid for a long time. Access tokens expire in the interval indicated, in minutes.

   Request to the server:

   ```
   curl --request POST --data
    "client_id=com.good.oauth2.client.general&grant_type=password&username=
   user@domain&password=account_password" https://login.blackberry.com/sso/oauth2/
   access_token
   ```

   Response from the server:

   ```
   {"scope":"com.good.oauth2.client.general goodCurUserLevel
    goodOrgID goodOrgRole goodOrgType inetUserStatus mail
    o","expires_in":600,"token_type":"Bearer","refresh_token":" TOKEN_VALUE
    ","access_token":" TOKEN_VALUE "}
   ```

2. You can use a refresh token to generate a new access token. Persist the refresh token in a secure storage location that is accessible to your app.

   Request to the server:

   ```
   curl --request POST --data
    "client_id=com.good.oauth2.client.general&grant_type=refresh_token&
   refresh_token=TOKEN_VALUE" https://login.blackberry.com/sso/oauth2/access_token
   ```

   Response from the server:

   ```
   {"scope":"com.good.oauth2.client.general goodCurUserLevel
    goodOrgID goodOrgRole goodOrgType inetUserStatus mail
    o","expires_in":600,"token_type":"Bearer","access_token":"TOKEN_VALUE"}
   ```

3. Exchange the access token for a JWT token at the BlackBerry Analytics portal.

   Request to the server:

   ```
   curl https://analytics.blackberry.com/enterpriseportal/api/v2/auth/oauth?
   accessToken=TOKEN_VALUE
   ```

   Response from the server:

   ```
   {"JWTToken":"JWT_TOKEN_VALUE"}
   ```

4. Make BlackBerry Analytics API calls with the JWT token. The JWT token expires after 1 hour. You can repeat steps 2 and 3 to get a new JWT token.

   Request to the server:

   ```
   curl -H 'Authorization:Bearer <JWT_TOKEN>'  https://analytics.blackberry.com/
   enterpriseportal/api/v2/os
   ```

Response from the server:


```
[{"versions":[{"name":"4.1.2"},{"name":"4.4.2"},{"name":"4.4.3"},
{"name":"4.4.4"},{"name":"5.0"},{"name":"5.0.1"},{"name":"5.0.2"},
{"name":"5.1"},{"name":"5.1.1"},{"name":"6.0"},{"name":"6.0.1"},{"name":"7.0"},
{"name":"7.1"},{"name":"7.1.1"},{"name":"7.1.2"},{"name":"8.0.0"},
{"name":"O"}],"osName":"Android"},{"versions":[{"name":"8.1.0"},
{"name":"9"},{"name":"p"}],"osName":"android"},{"versions":[{"name":"10.0"},
{"name":"10.0.1"},{"name":"10.0.2"},{"name":"10.0.3"},{"name":"10.1"},
{"name":"10.1.1"},{"name":"10.2"},{"name":"10.2.1"},{"name":"10.3"},
{"name":"10.3.1"},{"name":"10.3.2"},{"name":"10.3.3"},{"name":"11.0"},
{"name":"8.0.2"},{"name":"8.1.1"},{"name":"8.1.3"},{"name":"8.2"},
{"name":"8.3"},{"name":"8.4"},{"name":"8.4.1"},{"name":"9.0"},{"name":"9.0.1"},
{"name":"9.0.2"},{"name":"9.1"},{"name":"9.2"},{"name":"9.2.1"},{"name":"9.3"},
{"name":"9.3.1"},{"name":"9.3.2"},{"name":"9.3.3"},{"name":"9.3.4"},
{"name":"9.3.5"}],"osName":"iOS"},{"versions":[{"name":"11.0.1"},
{"name":"11.0.2"},{"name":"11.0.3"},{"name":"11.1"},{"name":"11.1.1"},
{"name":"11.1.2"},{"name":"11.2"},{"name":"11.2.1"},{"name":"11.2.2"},
{"name":"11.2.5"},{"name":"11.2.6"},{"name":"11.3"},{"name":"11.3.1"},
{"name":"11.4"},{"name":"11.4.1"},{"name":"12.0"}],"osName":"ios"}]
```

Sample apps

Visit Dynamics SDK Samples to view and download the following sample apps.

Sample app	Description
AppBasedCertImport	Demonstrates how to create an app that can import a user's PKI credentials using the BlackBerry Dynamics Certificate Credential Import API. For more information, see Certificate Credential Import and Creating user credential profiles for app-based certificates in the *UEM Administration Guide*.
AppKinetics	Demonstrates how to search for, create, and subscribe to client services (AppKinetics). The sample demonstrates these concepts by implementing a consumer and a provider for the Transfer File service. It is intended as a starting point for developers making use of BlackBerry Dynamics AppKinetics services. This app also demonstrates how to support multiple UIWindow objects.
AppKinetics save/edit client and server	Demonstrates how to write a client and a server that use the BlackBerry Dynamics Inter Container Communications API (also known as AppKinetics).
Bypass Unlock	Demonstrates how part of the application user interface can remain accessible after the BlackBerry Dynamics idle time out has expired. Bypass Unlock can be allowed or disallowed by implementing an application policy (see Adding custom policies for your app to the UEM management console). The ViewController that bypasses the unlock screen can be opened by pressing the volume controls if the application is in the foreground, or by using the auxiliary 'notifier' application (from the sub-project) if the application is in background.
Core Data	Demonstrates how to use BlackBerry Dynamics with a Core Data Incremental Data Store backed by an encrypted SQLite database. The sample shows how Core Data can be used to securely store a sample set of 20,000 fictional employee details and their office locations.
Crypto C	Demonstrates how to use the Crypto C language programming interface that allows an app to retrieve public key certificates that are stored in the BlackBerry Dynamics credentials store and use those certificates for signing and verification of messages and documents such as PDFs.
Greetings client and greetings server	Provides a client/server example of how to use the BlackBerry Dynamics Services API to communicate securely between two applications.
Remote DB	Demonstrates how to use BlackBerry Dynamics Remote Settings and DB APIs. The sample allows you to configure remote settings in the management console, and upon receipt, it stores them in the secure DB. Any changes to the settings are automatically synchronized and stored. This app also demonstrates the use of the GDiOS.executeBlock and GDiOS.executeUnblock APIs that can be used to locally block or unblock a user's access to the UI of a BlackBerry Dynamics app.

Sample app	Description
RSS Reader	Demonstrates how to use the BlackBerry Dynamics Secure Communications APIs to access resources behind the enterprise firewall. **Note:** If cellular access is enabled for a feed, the name of the feed is green in the app. If cellular access is not enabled for a feed, the name of the feed is black in the app.
RSS Reader Swift	Demonstrates how to use the BlackBerry Dynamics Secure Communication APIs for basic networking. It shows how to use `GET` with the `GDHttpRequest` API (with and without authentication) and the `GDNSURLSesssion` API. For session management, only `GDNSURLSesssion` sharedSession is supported.
Secure Storage	Demonstrates how to use the secure storage APIs, specifically the secure SQL database and secure file systems APIs.
Server-based Services	Provides a starting point for developers making use of BlackBerry Dynamics Server based Services. It retrieves the details of the google.timezone.service using getServiceProvidersFor. The sample prompts the user to supply a latitude and longitude and retrieves the corresponding results from Google. Before running this sample, make sure serviceId is created and associated with some existing app along with its endpoint details in the management console. Also, a user who is running this sample must have permission in the management console to run it. The Google timezone service is publicly available: • URL: https://maps.googleapis.com/maps/api/timezone/json?location=37,122&timestamp=85187&language=en&sensor=true • BlackBerry Service ID: com.blackberry.example.gdservice.time-zone • Latest Version: 1.0.0.0
Services Server Swift and Services Client Swift	Provides a complete client/server app that demonstrates how to implement a service with the Shared Services Framework. The service enables secure communication between two apps. The ServicesServerSwift app provides the service and works in conjunction with the ServicesClientSwift app, which calls the service. The samples also demonstrate how the file transfer service is used with the ServicesServeSwift app.
SwiftUI	The SwiftUI sample app pairs with Basic-iOS-Swift to provide examples of iOS apps before and after integrating BlackBerry Dynamics SDK. The two samples demonstrate features commonly used in BlackBerry Dynamics apps, including secure file storage, secure database, and secure communication (HTTP/S and Socket). The app is available on GitHub: https://github.com/blackberry/BlackBerry-Dynamics-iOS-Samples/tree/master/Dynamics-SwiftUI-Sample.

Testing and troubleshooting

This section provides guidance for testing and troubleshooting issues with your BlackBerry Dynamics apps.

Implementing automated testing for BlackBerry Dynamics apps

The BlackBerry Dynamics SDK includes the BlackBerry Dynamics Automated Test Support Library (ATSL) to support automated testing for your BlackBerry Dynamics apps. The library is delivered as a dynamic framework.

The library includes helper functions for testing common user interactions in BlackBerry Dynamics apps, such as activation and authorization. The configuration and structure of the library is compatible with the tools offered by Apple for user interface testing. It makes use of the following components:

- XCTest framework
- Accessibility identifiers

For more information about these components and iOS user interface testing, see Apple Documentation Archive: User Interface Testing.

You can use the BlackBerry Dynamics library and the native library components mentioned above to automate the building, execution, and reporting of your application tests.

Since the BlackBerry Dynamics ATSL is delivered as a framework, you cannot make your own changes to it. If you want to review the implementation and customize it, you can see the source in GitHub at https://github.com/blackberry/BlackBerry-Dynamics-iOS-Samples/tree/master/AutomatedTestSupportLibrary.

You can get the ATSL framework using Cocoa pods (see Configure a new or existing BlackBerry Dynamics app to use the dynamic framework).

Automated testing with the BlackBerry Dynamics sample apps

The sample apps included in the BlackBerry Dynamics SDK have been integrated with the BlackBerry Dynamics Automated Test Support Library (ATSL). Most Objective-C sample apps include test code that executes automated tests for provisioning and activation, and also demonstrate how to use the library to interact with generic UI components that are not related to BlackBerry Dynamics.

You can use the integration of the ATSL in any of the sample apps as a guide for integrating the library with your own BlackBerry Dynamics apps. Note the following details about ATSL integration:

Item	Description
Build target for testing	Each app has an `<AppName>UITests` build target that can be used by the XCTest framework. For more information about how to configure and run user interface tests and view results, see Apple Documentation Archive: Running Tests and Viewing Results.
Code for XCUITest tests	The test code is located in the sub-directory `<AppName>UITests`. The code is based on the XCTests framework. Most of the classes relate to user interface testing as `XCUIApplication`.

Item	Description
Use of ATSL for interaction with BlackBerry Dynamics screens	The test code uses helper functions in the ATSL. For example, the code calls the `loginOrProvisionBBDApp` method of the `BBDAutomatedTestSupport` class, which executes the entire activation process. You can validate ATSL helper function operations using XCTAssert macros. For example, this assertion will fail if activation fails: `XCTAssertTrue([ats loginOrProvisionBBDApp], @"Activation Failed!");`
Use of ATSL for general interactions	The test code also uses helper functions in the ATSL for general interactions, such as checking that an item exists in the user interface and then interacting with that item. The ATSL functions handle this by using XCTestCase and XCUIElement classes. These functions are defined in the category `XCTestCase +Expectations` and the group `Event_Helpers`.

Preparing for automated testing

As part of your automated testing you must activate a new installation of your BlackBerry Dynamics app or log in to an already activated app. This requires a user identifier (typically an email address), an access key, and a password. The ATSL can read activation or login credentials from a file in JSON format in the application resource bundle.

The file does not have any naming restrictions and must contain a single object with the following fields:

- GD_TEST_PROVISION_EMAIL: String that specifies the user's email address
- GD_TEST_PROVISION_ACCESS_KEY: String that specifies the access key
- GD_TEST_PROVISION_PASSWORD: String that specifies the user's password after provisioning or during login
- GD_TEST_UNLOCK_KEY: Specifies the unlock key, if necessary (optional)

For example:

```
{
      "GD_TEST_PROVISION_EMAIL": "user@acme.com",
      "GD_TEST_PROVISION_ ACCESS_KEY": "012345678901234",
      "GD_TEST_PROVISION_PASSWORD": "abcd"
}
```

There are different options to prepare the file:

- Manually generate an access key for a user in the management console, then manually edit the file.
- Manually generate and store a number of access keys in advance, then automatically edit the file and consume a key from the store.
- Automatically generate an access key, and optionally a user, with the BlackBerry Web Services APIs, then automatically edit the file.

Use the option that best suits your needs, based on your access to the management console and administrator level permissions, your familiarity with the BlackBerry Web Services APIs, and your preference for using a new activation for every test or running subsequent tests on the same app as an upgrade.

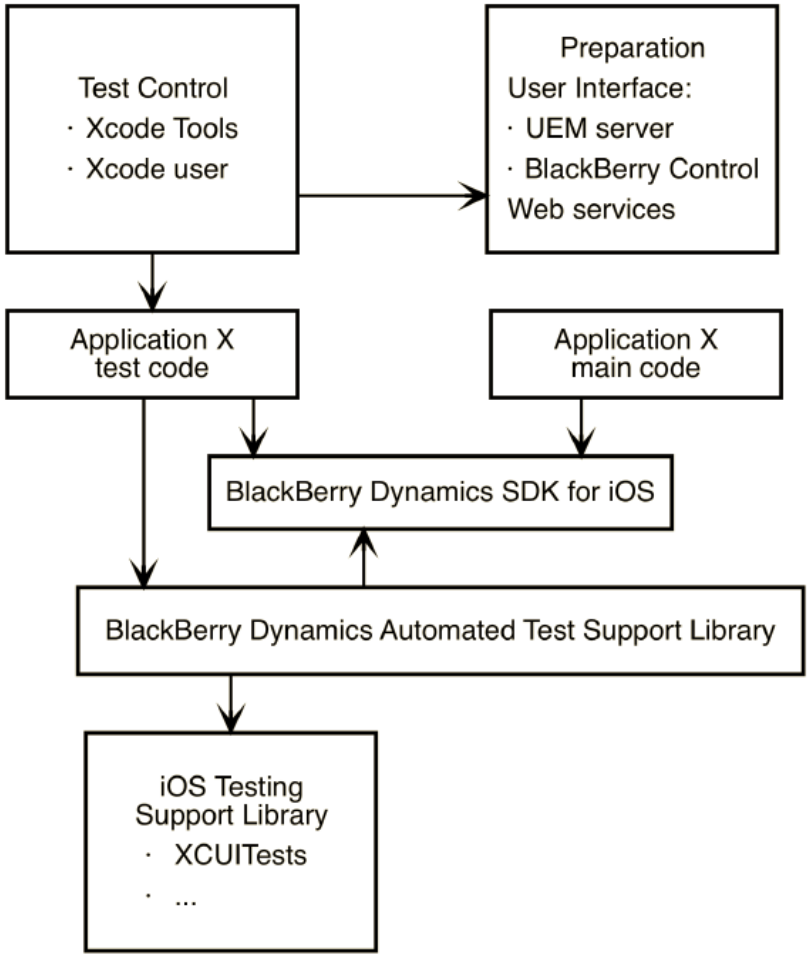
Please note the following:

- In the JSON file, the access key and unlock key have to be 15 characters and cannot have dashes.

- Automated testing works in Enterprise Simulation mode, but will have the same restrictions (for example, the app cannot connect to any back-end servers).
- There is currently an iOS known issue where automated tests can fail because the keyboard disappears occasionally in the simulator.
- In Xcode 8.3 and later, the XCTests framework executes tests in a random order, as a best practice. Tests should be self-contained and independent.
- Every BlackBerry Dynamics app needs a one-time initialization to complete activation. To ensure that this is executed first, define a dedicated test method that executes BlackBerry Dynamics activation and have that activation test executed in isolation.
- The source code is provided in Objective-C. If any of the app or test code is written in Swift, you must configure the code according to Apple standards for mixing Swift and Objective-C.

Components of a sample automated testing configuration

The following diagram illustrates the different components in a sample automated testing configuration:



Execute tests from the command line with Xcode tools

Follow these instructions to build the application and test code, run the tests on a connected iOS device or simulator, and record the results. You can run the command from a continuous integration system such as Jenkins.

Run the following command:

```
xcodebuild test -project "${XCODE_PROJ_PATH}" -scheme "${APP_SCHEME}" \
-sdk "${SDK_TYPE}" -configuration "${BUILD_TYPE}" \
CLANG_ALLOW_NON_MODULAR_INCLUDES_IN_FRAMEWORK_MODULES=YES \
DEVELOPMENT_TEAM="${TEAM_ID}" CODE_SIGN_IDENTITY="${CODE_SIGN}" \
FRAMEWORK_SEARCH_PATHS="${FRAMEWORKS_PATH}" \
HEADER_SEARCH_PATHS="${HEADERS_PATH}" \
-destination "${DESTINATION}" \
-only-testing:${APP_TARGET}/${CLASS}/${TESTCASE}
```

Define the following variables:

Variable	Value
XCODE_PROJ_PATH	The path to your project.
APP_SCHEME	The name of your app scheme.
SDK_TYPE	`iphoneos` or `iphonesimulator`.
BUILD_TYPE	`Release` or `Debug`.
TEAM_ID	The unique identifier of the team used for signing, as provided by an Apple developer program.
CODE_SIGN	`iPhone Distribution` or `iPhone Developer`.
FRAMEWORKS_PATH	The path to the folder containing the GD.framework package. The typical value is `<Dynamics_SDK_home_folder>/Frameworks`.
HEADERS_PATH	The path to the ATSL headers. This does not need to be specified if the ATSL is copied within the project folder.
DESTINATION	A string that describes the device that will be used to run the tests. For a physical device, it would be `platform=iOS,name=${DEVICE_NAME}`, where DEVICE_NAME is the name of the device as returned by `instruments -s devices`. For a simulator, it would be `platform=iOS Simulator,name=${SIMULATOR_NAME},OS=${IOS_VERSION}`, where IOS_VERSION is the version (for example, 10.3) and SIMULATOR_NAME is the name of the hardware simulator as returned by `instruments -s devices`.
APP_TARGET	The target for user interface tests.
CLASS	The test class to be executed. This value is optional. If it is omitted, all test classes in the target are executed.
TESTCASE	The method within the test class to be run. This value is optional. If it is omitted, all test methods in the class are run.

Test results are printed on standard output in the OCUnit format, and can be saved by a continuous integration system. You might need to convert the format to JUnit or another format that is suitable for your continuous integration system.

Execute tests from the Xcode IDE

The tests that you can run from the command line (Execute tests from the command line with Xcode tools) can also be run from within the Xcode IDE. This can be useful for investigating test failures with breakpoints or for running tests as you write the code.

The tests in the BlackBerry Dynamics sample applications for iOS can be run from within the IDE in the same way as any other tests that use the XCTests framework. Test results will be available in the Result view.

For more information, see Apple Documentation Archive: Running Tests and Viewing Results.

Add automated testing to your BlackBerry Dynamics iOS app

The following steps assume that the target app is already configured to use the BlackBerry Dynamics SDK.

1. If necessary, create a target in the project to run user interface tests. The target must have the type “iOS UI Testing Bundle”.
2. Add the BlackBerry Dynamics ATSL to the target for UI tests. Add the following to your applications podfile:

```
pod 'BlackBerryDynamicsAutomatedTestSupportLibrary', :podspec => 'https://
software.download.blackberry.com/repository/framework/dynamics/ios/
BlackBerryDynamicsAutomatedTestSupportLibrary.podspec'
```

3. If the test code is written in Swift, you must add the umbrella header to the bridging header, as required by Apple. Add "**#import <BlackBerryDynamicsAutomatedTestSupportLibrary/AutomatedTestSupportLibrary.h>**" to the bridging header.
4. Add or write code for your app tests. Use the helper functions in the ATSL in your test code.

You can use the code for the app tests in any of the sample apps as a starting point. The first app test, `testProvision`, executes BlackBerry Dynamics activation and unlock as an automated test.

Configure compliance settings so you can debug your app

Compliance profiles in BlackBerry UEM provide the ability to detect when a device OS is jailbroken and to initiate an enforcement action (this option is disabled by default). This feature extends to deployed BlackBerry Dynamics apps, compiled with SDK version 5.0 or later, where an active debugging tool is detected. Your options for configuring this feature depend on the version of BlackBerry UEM and the BlackBerry Dynamics SDK:

- If your organization uses BlackBerry UEM version 12.11 MR1 or later and the BlackBerry Dynamics SDK version 6.1 or later, when you enable the compliance setting to detect a jailbroken OS, you can configure the setting “Enable anti-debugging for BlackBerry Dynamics apps”. If enabled, the BlackBerry Dynamics Runtime stops a BlackBerry Dynamics app if it detects an active debugging tool. If disabled, the BlackBerry Dynamics Runtime takes no action when it detects an active debugging tool.
- In UEM versions earlier than 12.11 MR1, the “Enable anti-debugging for BlackBerry Dynamics apps” option is not present and this functionality is enabled by default. If you enable the compliance setting to detect a jailbroken OS, the BlackBerry Dynamics Runtime stops a BlackBerry Dynamics app when it detects an active debugging tool.

If you want to debug a BlackBerry Dynamics app in an environment where a compliance profile is applied, verify that the compliance settings are configured as required. Alternatively, you can use a non-debug build of your app to test it with the compliance settings enabled.

Using enterprise simulation mode

During the development and testing of your BlackBerry Dynamics app, you can run the app in enterprise simulation mode to verify its execution and functionality. This mode runs the app on an emulator (for example, the emulator that is supplied with the SDK). The user authentication process is simulated, so there is no direct communication with your organization's UEM server. As a result, a valid activation key is not required to activate the app.

Enterprise simulation mode is intended for development environments only. Note that even though there is no communication with the management server, communication with the BlackBerry Dynamics NOC still occurs during the initial activation of the app. The BlackBerry Dynamics NOC must be accessible from your testing environment.

You will notice the following differences when you run the app in enterprise simulation mode:

- A [Simulated] label appears in the BlackBerry Dynamics Runtime user interface.
- Any email address and activation key (PIN) is accepted for enterprise activation.
- The provisioning and policy setup flow is only simulated in the UI.
- A hard-coded set of profiles and policies from the management server is applied. Authentication delegation is not supported.

You should also note the following:

- If you run an app that was built for enterprise simulation mode on an actual device and not on an emulator, the app will be wiped.
- If you try to change to simulation mode when the app is already installed on a device, the app will be wiped.
- Lost password recovery is not available.
- Inter-container Communication (ICC) cannot be used; as a result, the Shared Services Framework cannot be used.
- The Secure Storage, Secure Communication, and Push Channel APIs (Android/iOS) are all available in enterprise simulation mode. The communication APIs will not be able to connect to your organization's application servers through the BlackBerry Dynamics proxy infrastructure. You can make connections to your organization's application servers if, for example, the AVD is running on a computer on your organization's LAN or VPN.

Enable enterprise simulation mode

1. In the Xcode project navigator, in the **Supporting Files** folder, open the associated **.plist** file.

   The prefix of the file includes the project name. For example, if the project name is FirstProject, the .plist file is named FirstProject-Info.plist. The file consists of a number of key-value pairs, including type.
2. Add a new row with the following values:
   - Key: **GDLibraryMode**
   - Type: **String (default)**
   - Value: **GDEnterpriseSimulation**

After you finish: To return to the default mode, delete the new row from the .plist file.

Troubleshooting common issues

Problem	Possible solution
Linking problems (for example, undefined symbols)	To verify that you included the necessary frameworks and libraries, check the appendix of the BlackBerry Dynamics SDK for iOS API Reference.
Access and password screens are poorly rendered	Verify that `GDAssets.bundle` has been added to the **Copy Bundle Resources** build phase and that the copy of the bundle distributed with the framework has been added to the project.
App crashes	• A crash in the early stages of the app lifecycle is most likely caused by a configuration issue. Review the errors printed in the logs for details. • Initialize the BlackBerry Dynamics libraries before calling any other BlackBerry Dynamics API.
Provisioning error	The email address or access key (PIN) has been incorrectly entered, or an old access key has been used. Double-check the credentials and, if necessary, send a new access key from the BlackBerry UEM management console.
Blank screen with keyboard when the app is launched	The BlackBerry Dynamics user interface is not correctly integrated with the app. Verify that `awakeFromNib` is implemented without warnings.
Blank screen after the password is entered	The app was authorized but failed to continue because the `GDAppEventAuthorized` event passed to `handleEvent` was not handled correctly.
A device command from BlackBerry UEM, such as wipe or lock, did not go into effect immediately	The app was unauthorized but failed to stop running because the `GDAppEventNotAuthorized` event passed to `handleEvent` was not handled correctly.

Logging and diagnostics

The processing activity of the BlackBerry Dynamics Runtime is logged by the runtime itself. The activity log is written to the BlackBerry Dynamics secure container on the device after deployment. You can configure how your BlackBerry Dynamics apps generate console log information. For more information about console logs controlled by developers and container logs controlled by UEM administrators, see the BlackBerry Dynamics Runtime activity log appendix in the API reference (Android/iOS).

If your app uses SDK version 5.0 or later, and the UEM administrator has turned off “Enable detailed logging for BlackBerry Dynamics apps” in the BlackBerry Dynamics profile (UEM), the app does not generate console log information. This provides additional protection against attacks by malicious users. This change has no impact on how container logs are generated.

The “Enable detailed logging for BlackBerry Dynamics apps” setting is off by default.

For BlackBerry Dynamics apps running SDK version 5.0 or later, console logs are generated only if this setting is turned on or if the app is running in enterprise simulation mode.

Configure logging for the Xcode console

The messages that are printed to the Xcode console can be configured in the build targets of the app project. Different targets can have different activity logging configurations. You can configure the logging to be detailed or selective. Changes to the console logging configuration take effect the next time the target is built and run, and do not affect the container log.

1. Open the app's **Info.plist** file.
2. Perform one of the following tasks:

Task	Steps
Configure detailed logging	**a.** Add a new row with the following values: • Key: **GDConsoleLogger** • Type: **String** • Value: **GDFilterNone** **b.** In Xcode 9.3 or later, to view the detailed logs, open the console app (Window > Devices and Simulators, or from OSX, Applications > Utilities > Console.app) and enable the following settings in the **Action** menu: • **Include Info Messages** • **Include Debug Messages**
Configure selective logging	**a.** Add a new row with the following values: • Key: **GDConsoleLogger** • Type: **Array** **b.** For each category that you want to omit from the console log, add a row under the logger row as an item in the logger's array. For each item, set the Type to **String** and the value to the desired logging level: **GDFilterDetailed**, **GDFilterInfo**, **GDFilterWarnings**, or **GDFilterErrors**.

Monitoring app log uploads by device users

You can use the GDLogManager class (Android/iOS) to monitor app log file uploads that are initiated by your organization's device users. This class does not manage or display information about log uploads that are initiated by the UEM administrator, or by management console profiles or policies.

GDLogManager provides the following information:

- Upload size
- Amount of data uploaded
- Events that indicate the following states:
  - Upload completed
  - Upload abandoned or canceled
  - Upload suspended
  - Upload resumed after suspension
- Actions for managing a log upload (cancel, suspend, resume)

- Enable detailed logging with detailedLoggingFor (Android/iOS)

When detailed logging is disabled by the management server profiles or policies, calling any API in the GDLogManager class has no effect. It is a best practice to check this setting using the getApplicationConfig API (Android/iOS). If detailed logging is enabled, present the log upload progress UI or any other related UI.

The following sample apps that are included in the BlackBerry Dynamics SDK for iOS demonstrate how to use GDLogManager:

- Objective-C samples: RSSReader, BypassUnlock, GreetingsClient, and GreetingsServer
- All Swift samples

Testing connectivity to application servers and diagnostic functions

You can use the GDDiagnostic class to test connectivity to application servers and other diagnostic functions.

The GreetingsClient sample app that is provided with the SDK demonstrates how to use the GDDiagnostic class.

Deploying your BlackBerry Dynamics app

Before you deploy your BlackBerry Dynamics app to your organization's production environment, you should test the app and the deployment process in a BlackBerry UEM environment that is reserved for development testing and evaluation. Coordinate with your organization's administrator to get access to a dedicated test environment.

BlackBerry Dynamics apps are fully supported for BlackBerry UEM. BlackBerry UEM is the recommended enterprise management solution to implement and use going forward, because it provides advanced app and user management features, advanced connectivity and networking options, expanded compliance and integrity checking, and the most recent BlackBerry Web Services REST APIs that your apps can leverage.

See the following resources for more information about distributing and managing your app in a BlackBerry UEM environment:

Task	Resource
Add your app to BlackBerry UEM and distribute it to users	See the following topics in the *BlackBerry UEM Administration Guide*: • Apps • Add an internal BlackBerry Dynamics app entitlement • Managing BlackBerry Dynamics apps
Configure BlackBerry Dynamics profiles that impact app functionality	See the following topics in the *BlackBerry UEM Administration Guide*: • Controlling BlackBerry Dynamics on users devices • BlackBerry Dynamics profile settings • BlackBerry Dynamics connectivity profile settings • Assigning profiles
Collect activity and compliance violation information for BlackBerry Dynamics apps	See the following topic in the *BlackBerry UEM Administration Guide*: • Activity and compliance violation reports for BlackBerry Dynamics apps

Configuring library version compliance

In a compliance profile or compliance policy, an administrator can enable the BlackBerry Dynamics library version verification compliance rule to specify an enforcement action if a BlackBerry Dynamics app is using a version of the BlackBerry Dynamics library that is not permitted. The available enforcement actions are "Do not allow BlackBerry Dynamics apps to run" and "Delete BlackBerry Dynamics app data."

In UEM, by default the BlackBerry Dynamics library version verification compliance rule is not selected and all versions are permitted. An administrator can enable this option and select specific versions to disallow.

Instructing users to trust the signing certificate

When a user opens a custom BlackBerry Dynamics app that has been deployed and installed on their device, the user is prompted to trust the organization that signed the app. An administrator should instruct users to complete this step. If a user does not trust the signing organization, the device will display a warning each time the user opens the app.

Deploying certificates to BlackBerry Dynamics apps

You can use any of the following options to deploy certificates to BlackBerry Dynamics apps. Each method requires configuration in the management console. Coordinate with your organization's administrator to select and configure the desired option.

Option	For more information
Personal Information Exchange files	See Using Personal Information Exchange files in this section.
CA certificate profile	See Sending CA certificates to devices and apps in the *UEM Administration Guide*.
User credential profile	See Sending client certificates to devices and apps using user credential profiles in the *UEM Administration Guide*.
SCEP profile	See Sending client certificates to devices and apps using SCEP in the *UEM Administration Guide*.
Shared certificate profile	See Sending the same client certificate to multiple devices in the *UEM Administration Guide*.

After certificates are distributed to a user's device, those certificates are shared and used by all of the BlackBerry Dynamics apps on the device. No additional programming is required by the app developer to support client certificates.

The management server and BlackBerry Dynamics apps also support the use of Kerberos for service authentication. For more information, see Using Kerberos in this section.

The SDK also provides a Crypto C language programming interface that allows an app to retrieve public key certificates that are stored in the BlackBerry Dynamics credentials store and use those certificates for signing and verification of messages and documents such as PDFs. Note that BlackBerry Infrastructure certificates cannot be retrieved from the store and that the private key will remain inaccessible. For more information, see the Crypto C Programming Interface appendix (Android/iOS) in the API reference.

Using Personal Information Exchange files

An organization can deploy corporate services that require two-way SSL/TLS authentication for users. A user is issued a password-protected Personal Information Exchange file (PKCS12 format, .p12 or .pfx) containing an SSL/TLS client certificate and a private key. This file can be provided to BlackBerry Dynamics apps to grant access to secure corporate services.

The BlackBerry Dynamics SDK supports the use of Personal Information Exchange files to authenticate BlackBerry Dynamics apps and to access secure services. All of the required operations to support client certificates are carried out by the BlackBerry Dynamics Runtime, with no additional programming required to handle the authentication challenge. For more information on how this is handled, refer to *HttpViewController.swift* in the Dynamics-iOS-Swift sample app. The app can use client certificates if:

- The app uses the BlackBerry Dynamics Secure Communication Networking APIs.
- The device user's UEM account is configured to support certificates.
- The certificates satisfy the certificate requirements.

After a user activates a BlackBerry Dynamics app, the app receives the Personal Information Exchange files. For each file, the user is prompted to provide the issued password so that the files and identification material can be installed. When this process is complete, the app can access the server resources that require two-way SSL/TLS authentication.

If more than one Personal Information Exchange file is required per user, the BlackBerry Dynamics Runtime selects the appropriate certificate using the following criteria:

1. Only client certificates that are suitable for SSL/TLS client authentication are eligible to send to the server. Certificates must have no Key Usage or Extended Key Usage, or Key Usage that contains “Digital Signature" or "Key Agreement”, or Extended Key Usage that contains "TLS Web Client Authentication”. Key Usages and Extended Key Usages must not contradict allowances for SSL/TLS client authentication.
2. If the server advertises the client certificate authority in the SSL/TLS handshake, only client certificates that have been issued by that authority are considered.
3. Expired certificates and certificates that are not yet valid cannot be selected.
4. If more than one certificate satisfies the above criteria, the BlackBerry Dynamics Runtime selects the most recently issued certificate.

Configuring support for client certificates

Certificate support is configured in the management console by the administrator. Contact your organization’s administrator to configure certificate support for BlackBerry Dynamics apps.

For more information about configuring certificate support in BlackBerry UEM, see the following:

- Manage settings for a BlackBerry Dynamics app in the *UEM Administration Guide*
- Sending certificates to devices using profiles in the *UEM Administration Guide*
- Connect BlackBerry UEM to a BlackBerry Dynamics PKI Connector in the *UEM Administration Guide*

Certificate requirements

- Client certificates must be in PKCS12 format, with the Certificate Authority (CA), public key, and private key in the same file.
- The PKCS12 file must have a .p12 or .pfx extension
- The PKCS12 file must be password-protected
- The source of the certificate can be your own internal CA, a well-known public CA, or an online tool such as OpenSSL or the Java keytool. You can use the following keytool example to generate a certificate, substituting your own values as required:

```
keytool -genkeypair -alias good123 -keystore good123.pfx -storepass good123 -
validity 365 -keyalg RSA -keysize 2048 -storetype pkcs12
```

- If the organization’s security policy uses FIPS standards, Personal Information Exchange files must be encrypted with FIPS-strength ciphers. If Personal Information Exchange files use a weak cipher, which is common for third-party applications when exporting identity material, you can use a tool like OpenSSL to re-encrypt the files with a FIPS-strength cipher. See the following example:

```
openssl pkcs12 -in weak.p12 -nodes -out decrypted.pem
       <enter password>
       openssl pkcs12 -export -in decrypted.pem -keypbe AES-128-CBC -certpbe
 AES-128-CBC -out strong.p12
      <enter password>
          rm decrypted.pem
```

Using Kerberos

BlackBerry Dynamics apps support both Kerberos PKINIT with PKI certificates and Kerberos Constrained Delegation. Kerberos PKINIT and Kerberos Constrained Delegation are distinct implementations of Kerberos. You can support one or the other for BlackBerry Dynamics apps, but not both.

With Kerberos PKINIT, authentication occurs directly between the BlackBerry Dynamics app and the Windows Key Distribution Center (KDC). User authentication is based on certificates that are issued by Microsoft Active Directory Certificate Services. No additional programming is required by the app developer to use Kerberos PKINIT.

With Kerberos Constrained Delegation, authentication is based on a trust relationship between the management server (BlackBerry UEM and a KDC. The management server communicates with the service on behalf of the app.

For more information about how to configure the desired Kerberos implementation in UEM, including requirements and prerequisites, see Configuring Kerberos for BlackBerry Dynamics apps in the *UEM Administration Guide*.

Legal notice

WERE FORESEEN OR UNFORESEEN, AND EVEN IF BLACKBERRY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW IN YOUR JURISDICTION, BLACKBERRY SHALL HAVE NO OTHER OBLIGATION, DUTY, OR LIABILITY WHATSOEVER IN CONTRACT, TORT, OR OTHERWISE TO YOU INCLUDING ANY LIABILITY FOR NEGLIGENCE OR STRICT LIABILITY.

THE LIMITATIONS, EXCLUSIONS, AND DISCLAIMERS HEREIN SHALL APPLY: (A) IRRESPECTIVE OF THE NATURE OF THE CAUSE OF ACTION, DEMAND, OR ACTION BY YOU INCLUDING BUT NOT LIMITED TO BREACH OF CONTRACT, NEGLIGENCE, TORT, STRICT LIABILITY OR ANY OTHER LEGAL THEORY AND SHALL SURVIVE A FUNDAMENTAL BREACH OR BREACHES OR THE FAILURE OF THE ESSENTIAL PURPOSE OF THIS AGREEMENT OR OF ANY REMEDY CONTAINED HEREIN; AND (B) TO BLACKBERRY AND ITS AFFILIATED COMPANIES, THEIR SUCCESSORS, ASSIGNS, AGENTS, SUPPLIERS (INCLUDING AIRTIME SERVICE PROVIDERS), AUTHORIZED BLACKBERRY DISTRIBUTORS (ALSO INCLUDING AIRTIME SERVICE PROVIDERS) AND THEIR RESPECTIVE DIRECTORS, EMPLOYEES, AND INDEPENDENT CONTRACTORS.

IN ADDITION TO THE LIMITATIONS AND EXCLUSIONS SET OUT ABOVE, IN NO EVENT SHALL ANY DIRECTOR, EMPLOYEE, AGENT, DISTRIBUTOR, SUPPLIER, INDEPENDENT CONTRACTOR OF BLACKBERRY OR ANY AFFILIATES OF BLACKBERRY HAVE ANY LIABILITY ARISING FROM OR RELATED TO THE DOCUMENTATION.

Prior to subscribing for, installing, or using any Third Party Products and Services, it is your responsibility to ensure that your airtime service provider has agreed to support all of their features. Some airtime service providers might not offer Internet browsing functionality with a subscription to the BlackBerry® Internet Service. Check with your service provider for availability, roaming arrangements, service plans and features. Installation or use of Third Party Products and Services with BlackBerry's products and services may require one or more patent, trademark, copyright, or other licenses in order to avoid infringement or violation of third party rights. You are solely responsible for determining whether to use Third Party Products and Services and if any third party licenses are required to do so. If required you are responsible for acquiring them. You should not install or use Third Party Products and Services until all necessary licenses have been acquired. Any Third Party Products and Services that are provided with BlackBerry's products and services are provided as a convenience to you and are provided "AS IS" with no express or implied conditions, endorsements, guarantees, representations, or warranties of any kind by BlackBerry and BlackBerry assumes no liability whatsoever, in relation thereto. Your use of Third Party Products and Services shall be governed by and subject to you agreeing to the terms of separate licenses and other agreements applicable thereto with third parties, except to the extent expressly covered by a license or other agreement with BlackBerry.

The terms of use of any BlackBerry product or service are set out in a separate license or other agreement with BlackBerry applicable thereto. NOTHING IN THIS DOCUMENTATION IS INTENDED TO SUPERSEDE ANY EXPRESS WRITTEN AGREEMENTS OR WARRANTIES PROVIDED BY BLACKBERRY FOR PORTIONS OF ANY BLACKBERRY PRODUCT OR SERVICE OTHER THAN THIS DOCUMENTATION.

BlackBerry Enterprise Software incorporates certain third-party software. The license and copyright information associated with this software is available at http://worldwide.blackberry.com/legal/thirdpartysoftware.jsp.

BlackBerry Limited
2200 University Avenue East
Waterloo, Ontario
Canada N2K 0A7

BlackBerry UK Limited
Ground Floor, The Pearce Building, West Street,
Maidenhead, Berkshire SL6 1RL
United Kingdom

Published in Canada