BlackBerry Dynamics SDK for iOS

Development Guide

5.0

Contents

About this guide

This guide is an introduction to the BlackBerry Dynamics SDK for iOS. It focuses on how to install the SDK, how to use the Xcode project template on the BlackBerry Developers for Enterprise Apps portal, and introduces the sample apps that are packaged with the SDK.

This guide is intended for software developers who already have an understanding of developing software for the iOS platform. It is not a basic tutorial.

For information about programming on iOS, see Start Developing iOS Apps on the Apple Developer site.

BlackBerry Dynamics background

The following sections provide some background information that can help you understand the features of the BlackBerry Dynamics SDK.

The way that these features are implemented in your environment will depend on how your administrator has configured your organization's servers, your network, and other infrastructure.

BlackBerry Dynamics API reference

The BlackBerry Dynamics SDK API reference describes the available interfaces, classes, methods, and much more.

You can access the iOS API reference online at https://developers.blackberry.com/us/en/resources/api-reference.html.

FIPS 140-2 compliance

BlackBerry Dynamics apps must comply with U.S. Federal Information Processing Standards (FIPS) 140-2. The BlackBerry Dynamics SDK distribution contains FIPS canisters and tools and, by default, enforces FIPS compliance.

There are two components involved in enabling FIPS:

Component	Description
BlackBerry Dynamics app	The app must start in FIPS-compliant mode. The BlackBerry Dynamics SDK determines whether a service is running in FIPS mode when the app communicates with the server to receive policies. All apps must be written for FIPS compliance.
Policy server (either standalone Good Control or BlackBerry UEM)	For more details on FIPS policies, see Readying your app for deployment: server setup.

FIPS compliance enforces the following constraints:

- MD4 and MD5 are prohibited. As a result, access to NTLM-protected or NTLM2-protected web pages and files is blocked.
- Wrapped apps are 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.

Easy Activation

The Easy Activation feature simplifies the provisioning process by allowing a BlackBerry Dynamics app to hand off activation to an app that is already installed on the device and can act as the activation delegate. The user has to retrieve and manually enter an access key only the first time the user installs a BlackBerry Dynamics app.

Securing cut-copy-paste on devices (Data Leakage Prevention, or DLP)

You can use the BlackBerry Dynamics SDK to protect certain data copied and pasted between apps on your users' devices.

For iOS, you don't need to do any additional programming to support secure cut and paste.

Server administrators must enable the **Data Leakage Prevention** policies in the management console.

To enable sharing among a group of apps, the apps must be provisioned from the same BlackBerry Control service for each user.

If the **Data Leakage Prevention** settings are enabled in your environment, you can work around them when you need to debug your app. For more information, see the BlackBerry Dynamics SDK API Reference.

Shared Services Framework

BlackBerry Dynamics-enabled apps can communicate with each other using the Shared Services Framework. There are two kinds of shared services:

- Server-side services
- Client-side services

The BlackBerry Dynamics SDK contains sample apps that show how these services work.

For a conceptual background, see BlackBerry Dynamics Services Framework.

Support for fingerprint authentication

Support for fingerprint recognition is a supplement to standard BlackBerry Dynamics secure user authentication, not a replacement for it. BlackBerry Dynamics includes the following policies related to fingerprint authentication. These settings are configured using policies in the management console:

- Allow or disallow fingerprint authentication for BlackBerry Dynamics-based apps in general.
- If fingerprint authentication is allowed, you can also allow or disallow it for BlackBerry Dynamics apps immediately after app coldstart. If you do not allow it after app coldstart, the user must enter the password for the app.
- Require the end user to enter a password after a specified interval.

Note: For app developers, no additional programming work is necessary for fingerprint authentication.

For more information, see BlackBerry Dynamics and Fingerprint Authentication.

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 in UEM or security policy in Good Control.

For applications built for iOS 11 or later, each application must add the NSFaceIDUsageDescription key to the Info.plist file. For more details about Face ID, see the BlackBerry Dynamics SDK for iOS API Reference.

If a BlackBerry Dynamics app is using version 4.0 of the SDK and the management console has not been upgraded to UEM 12.8 or later, or Good Control 5.0 or later, access to the Face ID feature is controlled by the Touch ID setting (Allow Touch ID for Idle Unlock).

Support for certificates

Type	Description
PKI certificates	BlackBerry Dynamics supports many popular uses of client-side Public Key Infrastructure (PKI) certificates to secure apps and communications: • General requirements for working with PKI certs • Description of client certificate sharing among BlackBerry Dynamics apps on a device • Kerberos PKINIT: Client certificates in the Kerberos authentication model. This is not Kerberos Constrained Delegation (KCD). For more information, see Details of support for client certificates.
SCEP	BlackBerry UEM version 12.10 and later support certificate enrollment using SCEP with Entrust and Microsoft NDES for BlackBerry Dynamics apps. Administrators can configure and assign a SCEP profile for BlackBerry Dynamics apps in the UEM management console. For more information, see “SCEP profile settings” in the UEM Administration Guide.

Support for the "Do not require password" setting

The BlackBerry Dynamics Runtime supports the "Do not require password" setting in a BlackBerry Dynamics profile in UEM or in a security policy in standalone Good Control. When this setting is enabled by an administrator, users cannot set a password for a BlackBerry Dynamics app or BlackBerry Dynamics container. Note that this setting does not apply to the device password.

This setting is available in BlackBerry UEM 12.7 or later and standalone Good Control 3.0.50.70 or later.

Security considerations

- Consider the security impact to your organization's environment before an administrator enables this setting. If enabling this feature does not meet security standards, consider other options, including authentication delegation or assigning the profile to specific users or groups that are already assigned device management profiles or other controls.
- Do not enable the "Do not require password" setting and authentication delegation in the same policy set.

- When the "Do not require password" setting is enabled, authentication can be accomplished only through user interaction or autonomously. For more information, see "canAuthorizeAutonomously" in the SDK programming reference for iOS or Android.

User experience when the rule is enabled or disabled

If a BlackBerry Dynamics app requires a password and the administrator enables the "Do not require password" setting, the next time the user opens the app, the app displays a message that a password is no longer required. As long as the feature is enabled, the user is not prompted for a password.

If the administrator disables the "Do not require password" setting, the next time the user opens the app, the app displays a message that a password is required. The user is prompted to specify a password.

iOS: Optional APIs for the "Do not require password" policy rule

You can call the `[GDiOS sharedInstance].canAuthorizeAutonomously` method to determine whether this feature is enabled for a BlackBerry Dynamics app.

If the app has received an APNS message or a Background Fetch period, or has been launched in the background, you can call the `[GDiOS sharedInstance].authorizeAutonomously` method to start the authorization process.

The SecureStore sample app illustrates the use of these methods.

For syntax and details, see the SDK programming reference.

Bypassing the App Lock screen

BlackBerry Dynamics supports the ability of an app to bypass the BlackBerry Dynamics user authentication/lock screen. Some organizations want this feature, particularly in VoIP apps where the user needs to respond quickly to an incoming call.

Note: Enabling this policy weakens the security inherent to BlackBerry Dynamics.

For information about requesting this feature, the necessary programming for bypassing the lock screen, the setup of a required app policy, and other details, see Bypass Unlock: BlackBerry Dynamics app Developer Guide.

BlackBerry Dynamics contributor code on GitHub

You can access contributed code samples for the BlackBerry Dynamics SDK on GitHub at https://github.com/jhawkinsatgood/gdcontributor. BlackBerry encourages you to take advantage of contributed code for the BlackBerry Dynamics SDK and to contribute your own code to share. The contributor code includes a number of complete BlackBerry Dynamics apps.

Supported languages

The BlackBerry Dynamics SDK supports the following languages. No SDK calls are required to use a particular language; the interface selects the appropriate language based on the language setting the user has configured on their device.

- English (US)

- Chinese (Simplified)
- Danish
- Dutch
- French
- German
- Italian
- Japanese
- Korean
- Portuguese (Brazil)
- Portuguese (Portugal)
- Spanish
- Swedish

Requirements

BlackBerry Dynamics software versions

- BlackBerry Dynamics SDK for iOS 5.0.x
- BlackBerry Dynamics Launcher Library for iOS 2.9.0.218

Compatibility with earlier releases

The latest release of the BlackBerry Dynamics SDK is compatible with the previous two releases.

Note: You should always build with and test against the most recent release. The most recent release has bug fixes and new features that you should test and deploy regularly.

Software requirements

iOS development

Item	Requirement
Deployment target	iOS 10 or later
iOS SDK	10.0 or later
Xcode	Xcode 9 or later Xcode 10 does not support docset archives for reference documentation. As a result, the BlackBerry Dynamics SDK for iOS does not include a docset archive.
Supported programming languages	• Objective-C • Swift 3 • Swift 4

BlackBerry Dynamics entitlement ID and version

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 for your apps, as well as for publishing and service provider registration. The BlackBerry Dynamics entitlement ID was formerly known as the app ID or GD App ID.

The entitlement ID is used in the app, in the BlackBerry UEM or standalone Good Control management console for app management, and in some administrative user interfaces on the application developer portal.

Note: The entitlement ID and entitlement version are different from the native application ID and native application version. The native application ID is a unique identifier for the app that is used by the OS and associated platforms (for example, the package name for Android or bundle identifier for iOS). The native application version is the app version number that you must change if you want to distribute a new version of an 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. For more information about when to change the entitlement ID and entitlement version, see the BlackBerry Dynamics API reference.

Requirements for the entitlement ID and entitlement version

Requirement	Description
Required for apps	You must define both the entitlement ID and the entitlement version for all your BlackBerry Dynamics apps, regardless of whether you use the BlackBerry Dynamics Shared Services Framework. Developers and administrators should ensure that the value specified for the `GDApplicationVersion` key in the app configuration files is the same as the value the administrator specifies in BlackBerry UEM or in standalone Good Control. The entitlement version is independent of any native version identifier. For more information, see Distinction from and use with native language identifiers.
Represent the same app across all platforms	The same entitlement ID must be used to represent the app across all platforms. By default, access to apps varies by the type of app: • By default, all versions of partner or ISV apps are available to all authorized users in any organization that the app has been published to. • By default, each version of a BlackBerry Dynamics app requires that the administrator grant access in BlackBerry UEM or in standalone Good Control before users can run the app on users' devices.
Naming scheme	Develop a naming scheme to meet your needs. For example: • Entitlement ID: com.manufacturingco.gd • Entitlement version: 1.0.0.0 • Native application version: 2.0
Entitlement ID format	• The general form of an entitlement ID is *<company_name>.<app_name>*. • The ID must use reverse domain name form, for example, com.company.example. Use a domain name owned by your organization. • The ID must not begin with com.blackberry, com.good, com.rim, or net.rim. • The ID can contain only lower-case letters, numeric digits, hyphens, and periods. • The string must follow the <subdomain> format defined in section 2.3.1 of RFC 1035, as amended by Section 2.1 of RFC RFC 1123.
Entitlement version value	• The value must use one to four segments of digits, separate by periods (x.x.x.x). • Each segment can be up to three digits and must not use a leading zero (for example, 01.02 is not valid). A segment can use a single 0. • The first release of an app should use the entitlement version 1.0.0.0.

Distinction from and use with native language identifiers

The Entitlement ID and Entitlement Version are BlackBerry Dynamics specific metadata and are independent of the identifiers needed by the app platforms themselves. The key point is that the values and the native language identifiers' values can be the same but they do not necessarily have to be. Listed below by platform are the equivalent native identifiers, which are where the values of Entitlement ID and version are stored.

- `Info.plist`
  - `CFBundleIdentifier`
  - `CFBundleVersion`

Unique native identifiers for enterprise apps

If you are developing a private app for use in your enterprise, make sure that the value you choose for the app's native identifiers (Bundle ID and others constructs used on other platforms) is unique, especially with respect to apps that are available through the public app stores.

Duplicate native identifiers can prevent the proper installation or upgrade of your own app.

For all your native identifiers, devise a naming scheme that you can be relatively certain is unique.

Mapping BlackBerry Dynamics entitlement ID to native identifiers

To take advantage of many features, such as Easy Activation, multi-authentication delegation, and the shared services framework, developers need to set up a map in the server between your defined Entitlement ID and the native identifiers on the platforms for which your app is distributed. The native platforms have no knowledge of the Entitlement ID; thus the mapping is needed for the operating systems to take over the actual function of the app.

Native version identifiers: * wildcard allowed for blocking app

The SDK supports use of native version identifiers in keeping with the conventions described by the major vendors. These same conventions apply to the use of the * wildcard in the server to deny apps by native version.

- Platform:iOS`CFbundleVersion`
- A series of integers separated by ".". No explicit limit on number of words.
- More information from Apple

The * character can be used in native version identifiers, but must always be preceded by a period (.) and must be the last character in the native version string. Examples:

- Allowed: 2.3.*
- Not allowed: 2.*.3
- 2.* includes 2.*.*

Requirements: Frameworks and libraries

The BlackBerry Dynamics runtime static library requires the following frameworks and libraries to be included in the Link Binary With Libraries build phase (not necessary for the BlackBerry Dynamics runtime dylib):

- CFNetwork.framework
- CoreData.framework
- CoreTelephony.framework

- GD.framework
- libz.dylib or libz.tbd
- LocalAuthentication.framework
- MessageUI.framework
- MobileCoreServices.framework
- Quartzcore.framework
- QuickLook.framework
- Security.framework
- SystemConfiguration.framework
- WebKit.framework

Requirement: URL scheme

Each application must add a custom URL scheme to the Info.plist file that consists of the bundle identifier with ".sc3" appended. Include the declaration in the same custom URL type declaration as the existing mandatory declarations (for example, the .sc2 scheme).

Required build-time declarations: URL type

Your app must declare a URL type so that it can be discovered by other apps on the same user's device. This enables BlackBerry Dynamics AppKinetics (shared services) communication, which is required for many BlackBerry Dynamics features.

The URL type and schemes are declared in an iOS app's Info.plist file.

The URL type must be the same as the application's native bundle identifier. Within the URL type declaration, the following URL schemes must be declared. For an example, if the native bundle identifier of the application is `com.example.gd.myapp` and its BlackBerry Dynamics app version is 1, then the declared URL type is `com.example.gd.myapp` and the schema declarations are shown below.

Table 1: URL type discovery schemes

Format	Description	Example
`com.good.gd.discovery`	Always required	Exactly as shown
`com.good.gd.discovery.enterprise`	Required for in-house apps developed by an enterprise; see Security changes in iOS 9 and later	Exactly as shown
`<native_app_identifier>.sc3`	Always required	`com.example.gd.myapp.sc3`
`<native_app_identifier>.sc2`	Enables an app to use authentication delegation and is required for all BlackBerry Dynamics apps	`com.example.gd.myapp.sc2`

Format	Description	Example
`<native_app_identifier>.sc2.<GD_app_version_#>`	Required only if your app provides a service that must be discoverable	`com.example.gd.myapp.sc2.1.0.0.0`

Security changes in iOS 9 and later

Apple introduced underlying security changes in iOS 9 that can impact BlackBerry Dynamics SDK apps, depending on how the apps are deployed. The changes relate to service discovery and keychain sharing. Review the new requirements below and modify your apps as necessary:

Requirement	Description
Add an "enterprise" discovery scheme	In the Xcode project's **Info.plist** file (or via the Xcode UI), in the **URL Schemes > CFBundleURLSchemes** section, add the following line: `com.good.gd.discovery.enterprise` This scheme is in addition to the other required or optional schemes discussed in Build-time required declarations: URL type.
Keychain group sharing for multiple apps	Apple's 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: 1. Open the app's project file. 2. Click the application target > **Capabilities** tab. 3. 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 below). 4. In the **Keychain Group** field, type `com.good.gd.data`. 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.
Common application identifier (App ID prefix)	An App ID prefix is a unique ID that groups a collections 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. **Note:** If your enterprise applications do not share the same App ID prefix, BlackBerry Dynamics Easy Activation does not work because the apps cannot discover each other. 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.

Using NSURLSystem for KCD

The BlackBerry Dynamics `GDHTTPRequest` method does not support Kerberos Constrained Delegation (KCD). If you plan to develop iOS apps for use with KCD, you can use the `NSURLSystem` provided by Apple instead.

GDAssets.bundle required in build phase

Ensure that `GDAssets.bundle` has been added to the **Copy Bundle Resources** build phase in Xcode and that the copy of the bundle distributed with the framework has been added to the project.

Compatibility with the BlackBerry Dynamics Launcher Library

The BlackBerry Dynamics SDK and the BlackBerry Dynamics Launcher Library are mutually dependent. If you use the BlackBerry Dynamics Launcher as a front-end for your apps, verify that you are using the latest compatible versions of the BlackBerry Dynamics SDK and the BlackBerry Dynamics Launcher Library. See BlackBerry Dynamics software versions for the required versions.

Note: Your apps may not run as expected if you do not use compatible versions of the BlackBerry Dynamics SDK and the BlackBerry Dynamics Launcher Library.

App UI restrictions

The BlackBerry Dynamics Runtime monitors the app UI to enforce a number of enterprise security policies configured in BlackBerry UEM or standalone Good Control. For example, a security policy may require the user to enter a password when the app transitions from the background to the foreground, or may lock the app UI after a certain period of inactivity.

The app UI must follow certain restrictions to enable monitoring by the BlackBerry Dynamics Runtime. For a complete explanation of the restrictions, open the BlackBerry Dynamics SDK API Reference for your platform and see the section “Application User Interface Restrictions” in the appendix.

Supported and unsupported features on iOS

Take into consideration these descriptions of supported or unsupported iOS features or constraints that might affect the design of your application or its use after it is deployed.

For information about other requirements, see Requirements.

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 4.0.0 and later supports Face ID. An administrator can enable or disable the feature in a BlackBerry Dynamics profile in UEM or security policy in Good Control. For applications built for iOS 11 or later, each application must add the NSFaceIDUsageDescription key to the Info.plist file. For more details about Face ID, see the BlackBerry Dynamics SDK for iOS API Reference.

Support for WKWebView

The BlackBerry Dynamics SDK for iOS version 4.2 and later supports secure WKWebView for displaying interactive web content. The SDK supports WKWebView for devices running iOS 11.0 or later.

Note the following support details:

- The SDK supports only one secure WKWebView instance at a time. The instance must be created programmatically.
- The supported versions of iOS require JavaScript injection by the BlackBerry Dynamics Runtime.
- The JavaScript Fetch API is not supported for iOS 11.2 and earlier.
- The JavaScript Fetch API is supported for iOS 11.3 and later, with the following known issue that will be addressed in a future release:
  - Requests with the raw file or blob binary object body content types do not work.

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
- Sending the following types of body data using XMLHttpRequest: ArrayBuffer, Blob, FormData, URLSearchParams, USVString

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

- Secure WKWebView creation from UIStoryBoard, including WKWebView controls that are added with the Xcode Interface Builder tool
- The JavaScript sendBeacon API
- The following Data Leakage Prevention (DLP) settings from BlackBerry UEM or standalone Good Control 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
- HTML5 persistent local storage
- HTML attributes for a link tag (for example, preconnect)
- WebSockets
- HTTP/2

Support for 64-bit ARM architecture

In addition to supporting 32-bit CPUs, the BlackBerry Dynamics SDK for iOS supports the 64-bit ARM (Apple A7) architecture.

Support for IPv6

In addition to IPv4, the BlackBerry Dynamics SDK supports Internet Protocol (IP) version 6.

BitCode not supported

BitCode is an intermediate, architecture-independent binary object format that allows developers to submit a "machine neutral" application to Apple for a final, architecture-dependent build.

However, the intermediate nature of BitCode is not compatible with the cryptographic requirements of the BlackBerry Dynamics SDK for iOS, 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 try to build an app using the BlackBerry Dynamics SDK for iOS and the directive `ENABLE_BITCODE=YES` you will see the following linker error:

```
Application has BitCode build option enabled. BlackBerry Dynamics SDK does not support BitCode. Please disable BitCode in your project Build Settings.
```

You must either remove the build directive `ENABLE_BITCODE=yes` or change it to `ENABLE_BITCODE=no`.

SiriKit not supported

SiriKit is an extension framework by Apple for integrating system voice control services with your application. If the SiriKit framework is integrated, the application's data could be sent out of its private space on the device and even off the device to servers operated by Apple.

Data sent via SiriKit integration cannot be protected by BlackBerry Dynamics . The mechanisms to send the data are private to Apple and cannot be secured by the BlackBerry Dynamics runtime on the device, nor can they be sent by BlackBerry Dynamics secure communication.

For these reasons BlackBerry Dynamics does not support SiriKit.

Support for the Apple Universal Clipboard

This release introduces support for 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 or standalone Good Control.

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

Link for FIPS in Objective-C or C++

Linking for FIPS is required. For background information, see FIPS 140-2 compliance.

If your application is written in Objective-C or C++, make the following project changes.

1. In the project's `default.xcconfig` file, add the following lines. (If you are not using an `xcconfig` file, add one and configure your build options to use it.) This ensures that the FIPS canister linking script `gd_fipsld` is called as part of the build process.
2. In your project build settings **Configurations**, make sure the **Debug** or other build configs point to the `default.xcconfig` you created in the step above.

```
FIPS_PACKAGE=$(CURRENT_ARCH).sdk
```

```
 LDPLUSPLUS=$(HOME)/Library/Application Support/BlackBerry/Good.platform/iOS/
FIPS_module/$FIPS_PACKAGE/bin/gd_fipsld
 LD=$(HOME)/Library/Application Support/BlackBerry/Good.platform/iOS/
FIPS_module/$FIPS_PACKAGE/bin/gd_fipsld
```

3. Ensure that the deployment target is set to the currently required version detailed in Software requirements.
4. Make sure that the `all_load linker` option is not specified. Check the **Other Linker Flags** field in the target build settings and remove this flag. Alternatively, you can use the **ObjC** linker flag.

Link for FIPS in Swift

If your project or app includes at least one Swift language source file, make the following project changes. Make sure that you set up linking for FIPS in Objective-C or C++. For more information, see Link for FIPS in Objective-C or C++. If you are not including source code in Objective-C, you must replace the contents of the **xcconfig** file, not add to it.

1. In the project's default **xcconfig** file, add the following line. If you are not using an xcconfig file, add one and configure your build options to use it.

   #

   ```
   FIPS_PACKAGE=$(CURRENT_ARCH).sdk
    LDPLUSPLUS=$(HOME)/Library/Application Support/BlackBerry/Good.platform/iOS/
   FIPS_module/$FIPS_PACKAGE/bin/gd_fipsld
    LD=$(HOME)/Library/Application Support/BlackBerry/Good.platform/iOS/
   FIPS_module/$FIPS_PACKAGE/bin/gd_fipsld
   ```

2. Ensure that the deployment target is set to the required version (see Software requirements).
3. In the project, go to **Target** > **> Build Settings** > **Linking** > **Other Linker Flags** and add the following:

   ```
   -B ${LDUTILDIR}
   ```

FIPS and 64-bit simulation

When you build for testing with the X86 64-bit simulator (the iPhone 6 simulated device), FIPS mode is not enforced, as it is when you build for the production architecture. Thus, you might perceive a difference in behavior with the simulator from the behavior in actual operation. BlackBerry recommends that you always test your app on actual iOS hardware and that you don't rely exclusively on the simulation.

Troubleshooting FIPS

If the app crashes it is probably because the FIPS canister has not been correctly linked to the application. If you check the log output you should see a line like this:

```
Crypto::initialise: FIPS_mode_set failed: fingerprint does not match
```

To correct this, make sure that you have followed the steps to build and link your application and that no errors have been reported.

If you see the following linker error when building for a device from Xcode

```
clang: error: invalid arch name '-arch armv7s'
```

For development or test, set **Build Active Architecture Only** to **Yes**.

For production release, be sure to set **Build Active Architecture Only** to **No**.

Steps to get started with the BlackBerry Dynamics SDK

Follow the steps below to start working with the BlackBerry Dynamics SDK. You can also see Getting Started with the BlackBerry Dynamics SDK for a step-by-step walkthrough of the stages of developing a BlackBerry Dynamics app.

1. Download and install the BlackBerry Dynamics SDK. See Downloading and installing the BlackBerry Dynamics SDK for iOS.
2. Familiarize yourself with the features of the BlackBerry Dynamics SDK.
   - See BlackBerry Dynamics background.
   - For more information about code samples, see Sample apps in Objective-C or BlackBerry Dynamics and the Swift programming language.
3. Understand the requirements and possible constraints on your programming.
4. Start working with the code. Do one of the following:
   - Create an Objective-C or Swift project with the Xcode template . The template includes all the necessary framework, required libraries, and the BlackBerry Dynamics SDK code for event handling and more.
   - If you have an existing project, you can add to it. See Manually add the BlackBerry Dynamics SDK event-handler skeleton in Objective-C, Add the BlackBerry Dynamics SDK event-handler skeleton using notifications in Objective-C , or Manually add the BlackBerry Dynamics SDK to your Swift project .
5. Become familiar with available classes and methods in the BlackBerry Dynamics API reference.
6. Add your own code.
7. Build your application.
8. Setup enterprise simulation mode or your own testing methods.
9. If necessary, troubleshoot your application. You might want to setup logging and diagnostics.
10. Ready your application for deployment.

Downloading and installing the BlackBerry Dynamics SDK for iOS

For information about software and hardware requirements, see Software requirements.

Download the BlackBerry Dynamics SDK

Visit BlackBerry Developers for Enterprise Apps to register and create an account. After you create an ccount, you can download the BlackBerry Dynamics SDK package and save the file on your computer.

Install the BlackBerry Dynamics SDK for iOS in the default location on macOS

The steps below install the BlackBerry Dynamics SDK in the default location. To change where the BlackBerry Dynamics SDK is installed, see Install the BlackBerry Dynamics SDK for iOS from a tar file.

Before you begin: Download the BlackBerry Dynamics SDK.

1. In the macOS GUI (not a shell), uncompress the downloaded file BlackBerry Dynamics SDK for iOS package.
2. Double-click the `sdk_version.pkg` installation wizard. Follow the instructions on the screen.
3. Read the introduction and confirm that your system meets the minimum requirements listed, then click **Continue**.
4. Read the Software License Agreement. Click **Continue** > **Agree**.
5. Click **Continue** to install the software on your computer or choose another network or peripheral drive if one is appropriate. If another location is available, it will appear as an option.

6. Do one of the following:
   - To use the standard installation, click **Install**.
   - To use the advanced installation, click **Customize**. If you use the advanced installation, you must install the Core SDK.

   For both installation types, you can omit the code examples, but including them is recommended.
7. Click **Install**.
8. Click **Close**.

The SDK is installed in the default location detailed in Location of installed artifacts

Location of installed artifacts

For consistency with other BlackBerry Dynamics products, to comply with Apple guidelines, and to make way for future cross-platform development the BlackBerry Dynamics SDK for iOS is now installed in the following default directories. The .pkg file installs in the following default locations.

- Main artifacts of the SDK are installed in the Library subdirectory in your home directory: `~/Library/Application Support/BlackBerry/Good.platform/iOS`
- A softlink to the `GD.framework` directory from the above is created in the system directory `/Library/Frameworks`
- The `GD.framework` directory is copied to the installed Xcode folder so that it is available for all iOS platforms. (This might break the integrity check in Xcode.)
- The Xcode template is in `~/Library/Developer/Xcode/Templates`

Install the BlackBerry Dynamics SDK for iOS from a tar file

You can install the BlackBerry Dynamics SDK from a tar archive into any directory you like.

Before you begin: Download the BlackBerry Dynamics SDK

1. Untar the file *`filename`*`.tar` into any directory you like, where *`filename`*`.tar` is the name of the BlackBerry Dynamics SDK archive that you downloaded.
2. To install the BlackBerry Dynamics SDK in your current working directory, run the following commands. The `install.sh` script first runs the uninstallation script.
   a. `cd Good.platform/iOS`
   b. `sudo ./install.sh`

The BlackBerry Dynamics SDK is installed in your current working directory.

After you finish: Adjust the paths in the `.xcconfig` file for your project to point to the `FIPS_module` folder in the main artifacts directory, wherever you installed it.

Uninstall the BlackBerry Dynamics SDK for iOS

Run the following commands from Terminal to remove all BlackBerry Dynamics installed artifacts, including the sample applications.

1. `cd` *`your_installation_directory`*
   The default location is `~/Library/Application Support/BlackBerry/Good.platform/iOS`.
2. `sudo ./uninstall.sh`

Create an Objective-C or Swift project with the Xcode template

You can use use the Xcode project template supplied with the BlackBerry Dynamics SDK to create a BlackBerry Dynamics app. The template includes a single view controller, a storyboard (or nib) file and includes all necessary frameworks and compile time directives.

Before you begin: Download the BlackBerry Dynamics SDK.

1. On your computer, start Xcode.
2. Select **New > Project**.
3. Follow the remaining prompts to finish creating the skeleton application.

All necessary frameworks and artifacts have been included in your stub.

Manually add the BlackBerry Dynamics SDK event-handler skeleton in Objective-C

You do not need to manually add the code described here if you use the supplied project template. The project template includes all this code automatically. See Create an Objective-C or Swift project with the Xcode template .

The application-delegate object `UIApplicationDelegate` manages the iOS app life cycle using events.

Many of the sample apps that are installed with the BlackBerry Dynamics SDK implement the event handling lifecycle. There are several key parts to implementing the life cycle of events. You can decide how to approach the life cycle of events. The following is one example.

- Specify the entitlement ID (also known as the app ID and version).
- Use `GDAppEvent` to process events.
- Special-case the `onNotAuthorized` events.
- On authorization, relying on `GDErrorNone` event, start the app UI.

This example is based loosely on Apple's Breadcrumbs sample app. However, there are significant differences in declarations and elsewhere. The important point is the event handling, not the overall purpose of the app.

1. Specify the entitlement ID and version. Add the following key value pairs to the application's `Info.plist`. For background, see BlackBerry Dynamics entitlement ID and version and Required build-time declarations: URL type.

   The values must correspond to the entitlement ID and version defined in standalone Good Control or BlackBerry UEM.
2. Import `GDiOS.h` and implement a skeleton `GDiOSDelegate` protocol. The example here relies on a boolean variable named `started`. An alternative approach is to declare the variable in your project's .m file.

```
// 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
```

3. Build the application and ensure there are no warnings due to partial implementation or errors.
4. Use `GDAppEvent` to process events. Move the application launch code from `didFinishLaunchingWithOptions` to `GDAppEvent` handler method.

```
// 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;
    }
}
```

5. Special-case the `onNotAuthorized` events. Make sure the application can handle problems such as authorization errors or functionality like remote wipe, a lockout, or blocking events. Implement these events in the `onNotAuthorized` function.

```
- (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;
    }
}
```

6. On authorization, start the application. Initialize and start the application UI with the `onAuthorized` function. The `GDErrorNone` event is returned by the BlackBerry Dynamics Runtime whenever a container is opened and no error has been encountered; you could test for it as an alternative to the boolean `started` shown here.

```
- (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;
    }
}
```

7. Enable FIPS linking. See Link for FIPS in Objective-C or C++.
8. Include the `GDAssets.bundle` in the build phase. See GDAssets.bundle required in build phase.

Add the BlackBerry Dynamics SDK event-handler skeleton using notifications in Objective-C

- Specify the entitlement ID (also known as the app ID and version).
- Use the `GDState` property of `GDiOS` to handle the state of the BlackBerry Dynamics Runtime.
- Special-case the `onNotAuthorized` events.
- On authorization, relying on the `GDErrorNone` event, start the app UI.

This example is based loosely on Apple's Breadcrumbs sample app. However, there are significant differences in declarations and elsewhere. The important point is the event handling, not the overall purpose of the app.

1. Specify the entitlement ID and version. Add the following key value pairs to the application's `Info.plist`. For background, see BlackBerry Dynamics entitlement ID and version and Required build-time declarations: URL type.

   The values must correspond to the entitlement ID and version defined in standalone Good Control or BlackBerry UEM.
2. Add the `BlackBerryDynamics` dictionary with 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`. The example here relies on a boolean variable named `'hasAuthorized'`. An alternative approach is to declare the variable in your project's .m file.

```
// 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
```

4. Build the application and ensure there are no warnings due to partial implementation or errors.
5. Make your `AppDelegate` an observer for state changing notifications.

```
// 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);
        }
    }
```

```
}
```

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

```
// 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;
        }
    }
```

7. Enable FIPS Linking (see Link for FIPS in Objective-C or C++).
8. Include the `GDAssets.bundle` in the build phase (see GDAssets.bundle required in build phase).

Programming with the BlackBerry Dynamics SDK and 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 BlackBerry Dynamics 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 or in standalone Good Control. 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 BlackBerry Dynamics entitlement ID and version). 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 doc 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. The API for that is `GDServiceType` in the `GDAndroid`, `GDiOS`, and `GDMac` classes.

Item	Description
Server cluster	The result of the service discovery query is an array of GDServiceProvider objects. Each object corresponds to a BlackBerry Dynamics entitlement identifier 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 BlackBerry Dynamics SDK for Android and for 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 connection succeeds, use that server. • If 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.

Sample apps in Objective-C

Visit Dynamics SDK Samples to view and download the following sample apps. In the SDK package, the samples are stored in ~/Library/ApplicationSupport/BlackBerry/Good.platform/iOS/Examples/objective-c.

Note: It is recommended to make a copy of the source code in another location before making any changes. Reinstalling or upgrading will overwrite or even remove the sample apps in the default location.

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 certifiactes" 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.
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 enterprise policy. This is implemented using the GD Application Policies feature (GD Application Policies are sometimes known as BlackBerry Dynamics Application-Specific Policies). 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.
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.
RSS Reader	Demonstrates how to use the BlackBerry Dynamics Secure Communications APIs to access resources behind the enterprise firewall.

Sample app	Description
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 a publicly available service: • 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

BlackBerry Dynamics and the Swift programming language

The BlackBerry Dynamics SDK for iOS is compatible with the Swift programming language from Apple (for supported versions, see Software requirements).

Note the following limitations and best practices:

Item	Description
Xcode template	The easiest way to use Swift with the BlackBerry Dynamics SDK is to create a new Xcode project with the Single View template that includes all necessary frameworks and compile-time directives. For more information about creating the template, see Create an Objective-C or Swift project with the Xcode template . You can include your Swift source files into the project and build with Xcode.
Sample apps	The BlackBerry Dynamics SDK includes the following sample apps in Swift. The sample apps are stored in ~/Library/Application Support/BlackBerry/Good.platform/iOS/Examples/. The Swift 3 versions are located in the "swift" folder, and the Swift 4 versions are located in the "swift4" folder. iPhone X is the default device for the Swift 4.0 sample applications. • RSSReaderSwift: Demonstrates how to use the BlackBerry Dynamics Secure Communication APIs for basic networking. The sample 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. • ServicesServerSwift and ServicesClientSwift: Provides a complete client/server app that demonstrates how to implement a service with the BlackBerry Dynamics 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.

Manually add the BlackBerry Dynamics SDK to your Swift project

Follow the steps below if you have an existing project. Otherwise, for new projects, consider using the BlackBerry Dynamics SDK Xcode template described in Create Objective-C or Swift project with BlackBerry Dynamics SDK Xcode template.

Before you begin: Download and install the BlackBerry Dynamics SDK.

1. In the **Link binary with binaries** section or **Build** section of your Xcode project, add the `GD.framework`.
2. Add the frameworks and libraries listed in Software requirements to the project.
3. In your code, import the BlackBerry Dynamics runtime module:

```
import GD.Runtime
```

4. Modify the `AppDelegate` class to implement the `GDiOSDelegate` protocol to implement a `handleEvent (anEvent:)` method, as shown here:

```
class AppDelegate: UIResponder, UIApplicationDelegate, GDiOSDelegate {
                    [...]
                    func handle(_ anEvent:GDAppEvent) {
                    [...]
                    }
             }
```

5. Add a variable to your `AppDelegate`:

```
var good: GDiOS?
```

6. Add the following code to the `didFinishLaunchingWithOptions` function:

```
self.good = GDiOS.sharedInstance() // Get BlackBerry Dynamics shared instance
 self.good!.delegate = self // Set delegate
 self.good!.authorize() // Authorize
```

7. Modify the app's **Info.plist** file to include the proper values for `GDApplicationID` and `GDApplicationVersion`, as described in BlackBerry Dynamics entitlement ID and version.
8. Define URL types with the same ID as the bundle ID, and URL schemes with the required discovery schemes detailed in Required build-time declarations: URL type (and any other discovery schemes required by your app).
9. Enable FIPS Linking (see Link for FIPS in Swift).
10. Include the GDAssets.bundle in the build phase (see GDAssets.bundle required in build phase).
11. Build your project.

Link for FIPS in Swift

If your project or app includes at least one Swift language source file, make the following project changes. Make sure that you set up linking for FIPS in Objective-C or C++. For more information, see Link for FIPS in Objective-C or C++. If you are not including source code in Objective-C, you must replace the contents of the **xcconfig** file, not add to it.

1. In the project's default **xcconfig** file, add the following line. If you are not using an xcconfig file, add one and configure your build options to use it.

   #

```
FIPS_PACKAGE=$(CURRENT_ARCH).sdk
 LDPLUSPLUS=$(HOME)/Library/Application Support/BlackBerry/Good.platform/iOS/
FIPS_module/$FIPS_PACKAGE/bin/gd_fipsld
 LD=$(HOME)/Library/Application Support/BlackBerry/Good.platform/iOS/
FIPS_module/$FIPS_PACKAGE/bin/gd_fipsld
```

2. Ensure that the deployment target is set to the required version (see Software requirements).
3. In the project, go to **Target** > **> Build Settings** > **Linking** > **Other Linker Flags** and add the following:

```
-B ${LDUTILDIR}
```

Testing and troubleshooting

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 source code.

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 Automated Test Support Library is delivered as source code, you can make your own changes to it. It is recommended that you make a copy of the source code before you make your own changes. Later releases of the SDK may include changes that you want to merge with your own.

The BlackBerry Dynamics ATSL is located in the sub-directory AutomatedTestSupport of the BlackBerry Dynamics SDK home directory. If you use the graphical installer to install the BlackBerry Dynamics SDK, the library is located under the BlackBerry Dynamics home directory at ~/Library/Application Support/BlackBerry/Good.platform/iOS/AutomatedTestSupport/. The library code is organized into a workspace file that also includes projects from every sample application.

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). Each sample app includes test code that executes automated tests of processing and behavior.

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

You must activate a new installation of a BlackBerry Dynamics app before you can run automated tests. This requires a user identifier (typically an email address), an access key, and a password. The ATSL can read activation credentials from any file in JSON format with the .json extension in the application resource bundle. Each sample app in the SDK includes a file named credentials.json in the test target.

For more information about the required content of the credentials file, see the `BBDAutomatedTestSupport` class in the reference documentation of the `fetchCredentialsFromFileAtPath:` method.

There are different options to prepare the file:

- Manually generate an access key for a user in the UEM or standalone Good Control 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.

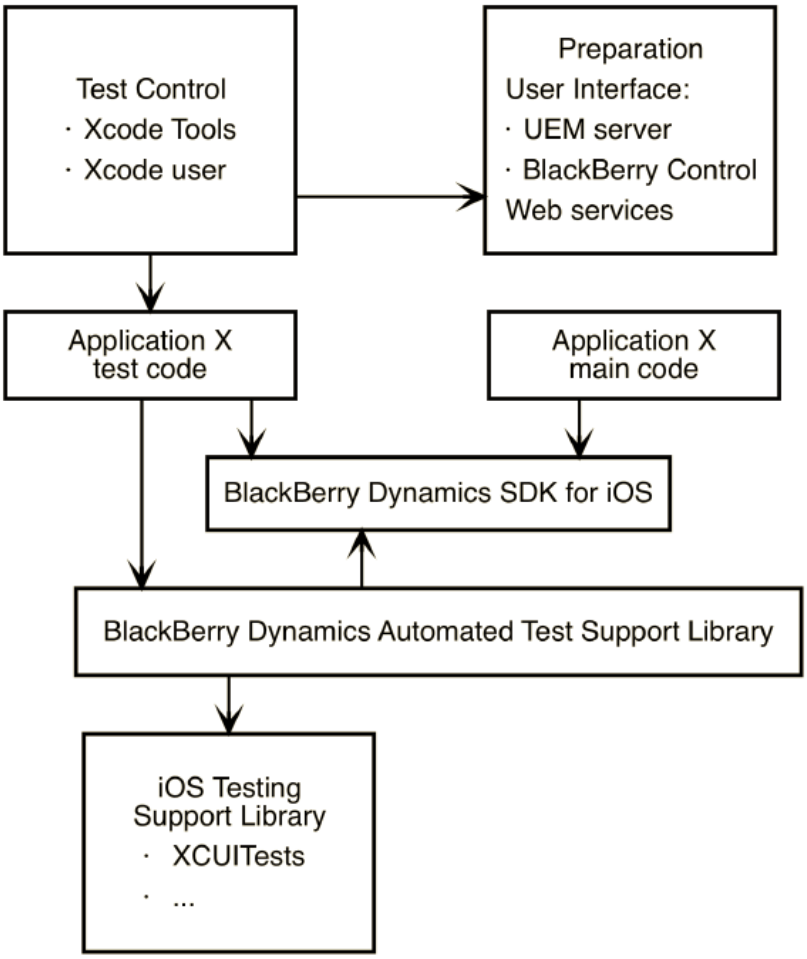
Please note the following:

- 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 ATSL to the test target.

   To import the BlackBerry Dynamics ATSL, add the following folder to the application project: BBD_SDK_HOME_FOLDER/AutomatedTestSupport/Core. Select the option to copy the ATSL folder into the project folder.
3. 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.

Disable compliance settings that check for a compromised OS

Compliance profiles in BlackBerry UEM and compliance policies in standalone Good Control provide the ability to detect when a device OS is jailbroken and to initiate an enforcement action. This feature extends to deployed BlackBerry Dynamics apps, compiled with SDK version 5.0 or later, where an active debugging tool is detected. If the detect jailbroken compliance setting is enabled, the BlackBerry Dynamics Runtime stops a BlackBerry Dynamics app if it detects an active debugging tool.

If you want to debug one of your BlackBerry Dynamics apps in an environment where a compliance profile or policy is applied, the compliance setting to detect jailbroken devices must be disabled. Alternatively, you can use a non-debug build of your app to test it with the compliance setting enabled.

The setting to detect jailbroken devices is disabled by default in UEM compliance profiles and enabled by default in standalone Good Control compliance policies.

Setup enterprise simulation mode

Enterprise Simulation mode in the BlackBerry Dynamics SDK facilitates app development work ahead of a full deployment to the BlackBerry Dynamics platform. It runs your app on an emulator (for example, one supplied with the iOS SDK) to efficiently verify new or changing functionality and performance.

Note: Enterprise Simulation mode should be used in development environments only. It is not intended for use in a production environment. In all cases, access to the BlackBerry Dynamics NOC is required.

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

   The prefix of the file you want reflects the project name. Thus, if you named your project **FirstProject**, for example, the `.plist` file is called `FirstProject-Info.plist`. The file consists of a number of key-value pairs, including type.

2. Add a new row with a two-finger tap (right-click) in the file and select **Add Row**.
3. Enter the following values in the new row:
   - Key: **GDLibraryMode**
   - Type: **String (default)**
   - Value: **GDEnterpriseSimulation**

Enterprise Simulation mode is now enabled for the app. If you need to return to the default authentication mode, simply delete the new row from the `.plist` file. Its absence means that the application will process default enterprise authentication, which requires a fully deployed BlackBerry Dynamics platform.

Run application in enterprise simulation mode

Running your application in simulation mode serves as a basic test of the BlackBerry Dynamics SDK installation.

Before you begin: See Setup enterprise simulation mode.

1. Open your Xcode project and run it. This builds the application and launches the device simulator.
   The Enterprise Activation screen appears.
2. Enter an email address.
   - Enterprise Simulation mode does not require a working address.
   - Any fictitious address is sufficient, including 'a@b.com.'
3. Enter an access key.
   Any sequence of 15 characters is acceptable.
4. Click **Next** or the play button.
   The onscreen progress-carousel displays the various activation steps.

   Note: In simulation mode no security policies are retrieved from the server during activation. Only default policies are applied.
5. When prompted by the **Security Password Setting** screen, enter any password that meets your server's password complexity requirements.
   In simulation mode, even the default security policy requires a password.
6. Click the **Next** button to set the password and see the simulator's display

Running the skeleton application in Enterprise Simulation mode verifies that the SDK was installed correctly and that a connection to the BlackBerry Dynamics infrastructure is available.

Be aware that, when you are running a simulation, the BlackBerry Dynamics runtime makes no attempt to contact the policy server, so there will be no user authentication at the enterprise level and therefore no communication through the enterprise firewall. Depending on the version of the SDK you are using, if you build the application in simulation mode and attempt to then run it on a device, you will receive a message to the following effect:

This application is in simulation mode and cannot be run on device

or

Application Access Denied, Wipe/Block NEM Mode

Additional connectivity restrictions and limitations in Enterprise Simulation mode include:

- Provisioning and policy set-up flow are merely simulated in the UI.
- Hardcoded set of security and compliance policies in operation; authentication is not delegated.
- If run on a real device, not a simulator, the application will be wiped.
- Any attempt to change the mode of an application that is already installed from default to simulated will also result in a wipe.

- There is no lost password recovery.
- BlackBerry Dynamics Inter-Container Communication (ICC) cannot be used. This means that the BlackBerry Dynamics Shared Services Framework cannot be used.

.

Troubleshooting

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	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 or Good Control administrators, see BlackBerry Dynamics Runtime activity log.

If your app uses SDK version 5.0 or later, and the UEM or Good Control administrator has turned off “Enable detailed logging for BlackBerry Dynamics apps” in the BlackBerry Dynamics profile (UEM) or security policy (Good Control), 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.

Log message categories

Messages in the activity log are assigned to one of four categories:

- Errors: critical failures
- Warnings: failures that arise but from which the BlackBerry Dynamics Runtime has recovered
- Info: normal operational activity
- Detailed: additional diagnostic information used for troubleshooting complex problems

You can configure the logging system to filter some or all messages. By default, only messages that belong to the Errors, Warnings, and Info categories are printed.

The logging locations on the IDE console and device container are configured differently and independently. In principle, you control the console log and the BlackBerry UEM administrator controls the container log.

Configure detailed logging for the Xcode console

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

1. Open the app's `Info.plist` file.
2. Add a row with `GDConsoleLogger` as its key.
3. Set `String` as the type of the new row.
4. Set `GDFilterNone` as the value of the new row.

After you finish: In Xcode 9.3 or later, to view the detailed logs you must open the console app (from Xcode go to Window > Devices and Simulators, or from OSX go to Applications > Utilities > Console.app) and enable the following settings in the Action menu:

- Include Info Messages
- Include Debug Messages

Configure selective logging for the Xcode console

If there are multiple `Info.plist` files, check that you are editing the correct one by opening the Info tab of the app target being built to make sure the new settings are there.

As with detailed logging, the console will only include log messages that are not in any of the categories specified in the `Info.plist`.

To set the target's activity logging configuration to print a selection of message categories, do the following:

1. Open the app `Info.plist` file.
2. Add a row with GDConsoleLogger as its key or change the existing row as follows:
   a) Set `Array` as the type of the logger row.
   b) For each category you don't want to include, add a row under the logger row as an item in the logger's array.
   c) Set the item to type `String`.
   d) Set the value to one of the following:
      - GDFilterDetailed
      - GDFilterInfo

- GDFilterWarnings
- GDFilterErrors

Configure logging in Good Control

You can specify the message categories that are recorded in the container log file in the Good Control management console. To configure logging, you must be a Good Control administrator. Container logging can be set for a particular installation of the app provisioned for a specific user. Selective logging for the container is not supported.

Changes that you make to container logging are effective immediately if the app is running and connected to the BlackBerry Dynamics infrastructure. Otherwise, changes are effective as soon as app does connect. Changes to container logging have no effect on the Xcode logging.

The app can export or upload the container log file using the BlackBerry Dynamics API. For more information, see the function definitions in the `GDFileSystem` class reference in the BlackBerry Dynamics SDK for iOS API Reference.

GDLogManager class for log uploading

You can use the `GDLogManager` class that is included with the BlackBerry Dynamics SDK to monitor app log file uploads that are initiated by end users. This class does not manage or display information about log uploads initiated by the BlackBerry UEM administrator's policies or explicit actions.

The following samples that are delivered with the SDK illustrate how to use `GDLogManager`:

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

`GDLogManager`'s displayed information and functions include the following:

- Size of whole upload
- Amount of data uploaded so far
- 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 upload
  - Suspend upload
  - Resume upload

For more information, see `GDLogManager` in the BlackBerry Dynamics API reference.

When detailed logging is disabled by policy, calling any API in the `GDLogManager` class has no effect. It is a best practice to check the setting of this policy in the app configuration using the `getApplicationConfig` API. If detailed logging is enabled, present the log upload progress UI or any other related UI.

GDDiagnostic API

The BlackBerry Dynamics SDK includes an API that you can use to test connectivity to application servers and other diagnostic functions. See the discussion of the `GDDiagnostic` class in the BlackBerry Dynamics API reference.

The GreetingsClient sample is delivered with the SDK to illustrate how to use `GDDiagnostic`.

Readying your app for deployment: server setup

You want to test your app on a BlackBerry server before you deploy it into production. You need to become familiar with how to setup and configure such a server. You have options available: either BlackBerry UEM or the older Good Control.

Check with your IT or other department to see what test servers might already be available in your organization.

BlackBerry UEM preferred

BlackBerry UEM is the primary server configuration to test and deploy your app.

If you upgrade from Good Control to BlackBerry UEM, you not only get to use the great feature set that Good Control provides but you also get to take advantage of an enhanced feature set such as:

- Support for more policies for operating systems
- Better app management
- More container types
- Improved administration and provisioning
- Advanced connectivity and networking
- Expanded compliance and integrity checking
- Additional email, content, location, and certificate features
- Access to BlackBerry Web Services APIs

For information on how to use BlackBerry UEM to manage BlackBerry Dynamics apps, see the BlackBerry UEM Administration Guide.

Standalone Good Control

The standalone Good Control server is also available, but BlackBerry encourages you to use BlackBerry UEM for your tests and deployment. For information about getting started with Good Control, see Developer Bootstrap: Good Control Essentials.

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 previous releases of UEM, standalone Good Control, and the SDK, BlackBerry Dynamics apps could be blocked or deleted unintentionally if an app used a version of the library that was not available in the version list (for example, a pre-release version of the SDK).

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.

In standalone Good Control 5.0, the following options have been added to this compliance rule:

- Allow all BlackBerry Dynamics library versions: Apps that use any version of the SDK library are allowed. If this option is enabled, the administrator cannot select specific versions to allow or disallow. By default, this option is disabled.

- Allow unlisted BlackBerry Dynamics library versions: Apps that use versions of the SDK library that are newer than the latest version listed in the compliance rule are allowed. The administrator can still allow or disallow specific versions of the library from the version list. By default, this option is enabled.

Consult with the UEM or Good Control administrator to ensure that the compliance rule is configured appropriately.

iOS requires users trust your app's signing certificate

When your end users launch an app deployed by your organization (any app that is not distributed through the App Store), they must trust the organization that signed the app. Otherwise, if users do not trust the signing organization, the iOS device displays a warning each time a user opens the app.

This is a feature of iOS.

Details of support for client certificates

BlackBerry Dynamics SDK support for personal certificates (PKCS12 or PKI certs)

The BlackBerry Dynamics SDK has been enhanced to support personal certificates for authentication of applications at runtime.

No programming is required by the BlackBerry developer on any of the client BlackBerry Dynamics SDK platforms to take advantage of this feature. All operations are carried about by the BlackBerry Dynamics Runtime. The app must use the BlackBerry Dynamics Secure Communication Networking APIs provided in prior releases, the employee's account must be correctly configured, and the GC must be the 2.0.xx.yy release later.

An enterprise can deploy corporate services requiring two-way SSL/TLS mutual authentication in order to authenticate their employees. Through the enterprise, the employee may be issued or otherwise obtain a password protected Personal Information Exchange file (PKCS12/p12/pfx) containing a SSL/TLS client certificate and private key required by such services for authentication purposes. This file may be installed on various machines and devices, including BlackBerry Dynamics apps, so that access can be granted to these services.

Setup in Good Control

Requirements of the certificates themselves are described in Certificate requirements and troubleshooting.

To deploy Personal Information Exchange files with BlackBerry Dynamics apps, the following steps must be taken to configure the GC and employee's account. For more information, see the Good Control and Good Proxy Admin Help.

- After the GC is installed, an administrator may choose to extend the default 24-hour period that an employee's protected Personal Information Exchange file shall be cached by the GC server.
- An administrator must add all BlackBerry Dynamics apps that access services requiring client authentication to the **Certificates -> App Usage** tab,
- An administrator must enable **Use PKCS12 Certificate Management** in the employee's security policy,
- An administrator or employee must upload their Personal Information Exchange files to the **Certificates** tab.

Behavior of personal certificates in the app

After the employee activates a BlackBerry Dynamics app enabled for access to server resources requiring client authentication, it receives their Personal Information Exchange files, provided they are still cached on the GC. For each file, the employee is asked to enter their password protecting the file contents, so the identification material can be installed. Once installed, provided the identification is correct, the BlackBerry Dynamics app is granted access to server resources requiring two-way SSL/TLS mutual authentication when connecting.

If there is more than one Personal Information Exchange file required per employee, the BlackBerry Dynamics Runtime ensures that the certificate chosen to send to the server meets all of the following criteria:

1. Only client certificates suitable for SSL/TLS client authentication are eligible for sending to the server. That is, certificates that have no Key Usage and Extended Key Usage, or Key Usage contains "Digital Signature" or "Key Agreement", or Extended Key Usage contain "TLS Web Client Authentication", and those whose Key Usages and Extended Key Usages do not contradict allowances for SSL/TLS client auth.
2. If the server advertises the client certificate authority in the SSL/TLS handshake, only client certificates issued by these authorities will be considered
3. Only current client certificates will be considered (that is, certificates that have not expired or are not yet valid)

Usually this is sufficient to identify the correct client certificate, but if there is still more than one certificate meeting all of the above criteria then the first one is used. If the certificate chosen is not the desired one, the administrator or employee can manage this by removing the undesired client certificate from Good Control. The administrator can also increase the chance of success by ensuring the server is configured to advertise the client certificate authority in the SSL/TLS handshake.

Certificate requirements and troubleshooting

Make sure your certificates conform to these requirements:

- Certificates must be in PKCS 12 format: Certificate Authority (CA), public key, and private key, all in the same file.
- The PKCS12 file must end with the extension `.p12` or `.pfx`.
- The PKCS 12 file must be password-protected.

There are many sources of certificates:

- Your own internal certification authority (CA)
- A well-known public CA
- Tools from the Internet, such as OpenSSL's keytool command. For example, the following is sufficient to generate a PKCS 12 certificate that is usable with Good Control; substitute your own values for alias the keystore name and the keystore password. If in doubt consult information on the Internet about all the possible options on the keytool command:

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

Beware of weak ciphers from export

Personal Information Exchange files are encrypted, and therefore must be encrypted with FIPS-strength ciphers if to be used when FIPS is enabled on the employee's security policy.

For their own maximum interoperability with other systems, it is common for third-party applications, for example the macOS keychain, to export identity material (credentials) using weak ciphers.

The administrator or employee can use a tool such as the OpenSSL command line to re-encrypt the file with a FIPS-strength cipher like so, which re-encrypts with the AES-128-CBC cipher:

```
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
```

Client certificate sharing among BlackBerry Dynamics-based applications

The BlackBerry Dynamics SDK supports the "sharing" of a single client certificate among all BlackBerry Dynamics-based applications for an end-user.

If the security policy for authentication via client certificates is enabled in Good Control or UEM and one or more client certificates have been uploaded to the server, those certificates are used for user authentication by all BlackBerry Dynamics-based applications on the user's device.

- No programming is required.
- Client certificates must be enabled in Good Control or UEM and at least one PKCS 12 certificate for a user must be uploaded to server.

Kerberos PKINIT: User authentication with PKI certificates

The BlackBerry Dynamics SDK supports Kerberos PKINIT for user authentication using PKI certificates.

No programming is required to use Kerberos PKINIT.

Important: Kerberos PKINIT is distinct from Kerberos Constrained Delegation (KCD). PKINIT relies on the Key Distribution Center (KDC), which should not be confused with "KCD".

Kerberos PKINIT	Kerberos Constrained Delegation
Kerberos PKINIT authentication is between the BlackBerry Dynamics app and the Windows Key Distribution Center (KDC), which communicate directly, and user authentication is based on certificates issued by Microsoft Active Directory Certificate Services.	**Note:** For PKINIT, Kerberos Constrained Delegation must not be enabled. If Kerberos Constrained Delegation has been configured, a BlackBerry Dynamics app does not use Kerberos PKINIT to access the defined KCD realms. Instead, when Kerberos Constrained Delegation is used, a trust relation has been previously established between BlackBerry Control and the Key Distribution Center, and BlackBerry Control communicates with the service on behalf of the app. Kerberos Constrained Delegation takes precedence over Kerberos PKINIT, even if the user has a valid certificate.

Key requirements for PKINIT

Organizations that want to use Kerberos for BlackBerry Dynamics apps must make sure the following requirements are met.

Servers

- Kerberos Constrained Delegation must not be enabled.
- Windows Key Distribution Center (KDC) services for KDC server certificates issued by a Certificate Authority (CA) via the Active Directory Certificate Services must come only from the following Windows Server versions. No other server versions are supported.
  - Internet Information Server with Windows Server 2008 R2
  - Internet Information Server with Windows Server 2012 R2
- In BlackBerry Control:
  - The KDC hosts must be in the Allowed Domains of the Connectivity Profile applied to the affected users' policy sets.
  - Valid KDC service certificates must be located either in the BlackBerry Dynamics Certificate Store or the Device Certificate Store.

Client certificates

- The minimum keylength for the certificates must be 2,048 bytes.
- Client certificates must include the User Principal Name (UPN) (for example, user@domain.com) in the Subject Alternative Name (SAN) of object ID (OID) szOID_NT_PRINCIPAL_NAME 1.3.6.1.4.1.311.20.2.3, as specified by Microsoft. See Microsoft Support: Object IDs associated with Microsoft cryptography.
- The domain of the UPN must match the name of the realm of the Windows Key Distribution Center (KDC) service.
- The Extended Key Usage (EKU) property of the certificate must be Microsoft Smart Card logon (1.3.6.1.4.1.311.20.2.2).
- Certificates must be valid. Validate them against the servers listed above.

Client applications

- In BlackBerry Work, to allow the use of client certificates, you must enable the `useEASAuthCert` setting.
- Apps must not send any password in the HTTP/HTTPS request.
- Apps must either set the HTTP/HTTPS header `WWW-Authenticate: Negotiate`, or not set any authorization method in the HTTP or HTTPS request, to which the server has responded with 401 `WWWAuthenticate: Negotiate`. For more information, see SPNEGO-based Kerberos and NTLM HTTP Authentication in Microsoft Windows.

Key points

The following are key points to note when integrating BlackBerry Dynamics and Kerberos infrastructure:

- The KDC host must be in the Allowed Domains of the Connectivity Profile applied to the affected users' policy sets in BlackBerry Control.
- The KDC host must be listening on TCP port 88 (Kerberos default port).
- BlackBerry Dynamics does not support KDC over UDP.
- BlackBerry Dynamics does not use Domain Name System (DNS) records such as SRV, `CNAME`, or `TXT` to locate the correct KDC. That is, the KDC must have an `A` record (IPv4) or `AAAA` record (IPv6) in your DNS.
- BlackBerry Dynamics does not use Kerberos configuration files (such as `krb5.conf`) to locate the correct KDC.
- The KDC can refer the client to another KDC host. BlackBerry Dynamics will follow the referral, as long as the KDC host that is referred to can be reached by BlackBerry Dynamics. This setting is defined in the **Allowed Domains** of the Connectivity Profile that is applied to the affected users' policy sets in BlackBerry Control.
- The KDC can obtain the TGT transparently to BlackBerry Dynamics from another KDC host.

Background on PKINIT, with FAQ

Consider the interactions in this KDC diagram.

Kerberos PKINIT authentication requires the client (in the drawing, the human John, running a BlackBerry Dynamics-enabled application) to be able to contact:

- When initializing the user session, the user's Key Distribution Center (KDC) Authentication Service (AS) to obtain a Ticket-Granting Ticket (TGT)
- When establishing a connection to a resource (in the drawing, Service "A"), the resource's KDC Ticket-Granting Service (TGS)

In a large organization users and resources might belong to various realms and there may be many KDCs, so how does BlackBerry Dynamics find the right one?

1. How does the client locate the user's KDC Authentication Service when initializing the user's session?
   - Password-based authentication

     The realm in the user name must contain the host name of the KDC AS. For example:

     User: `user@MY.REALM.COM`

     Password: `myPassword`
   - Certificate-based authentication: This is PKINIT.

     The realm in the UPN of the user's certificate must contain the host name of the KDC AS. For example:

     `UPN (OID 1.3.6.1.4.1.311.20.2.3): user@MY.REALM.COM`
2. How does the client locate the resource's KDC Ticket-Granting Service (TGS) when retrieving the resource?

   BlackBerry Dynamics attempts to obtain a TGS from the host in the domain of the resources URL. For example:

   URL: http://resource.myrealm.com/index.html

   The client will connect to KDC TGS running on host myrealm.com on TCP port 88.

Legal notice

SERVICES, DOWNTIME COSTS, LOSS OF THE USE OF BLACKBERRY PRODUCTS OR SERVICES OR ANY PORTION THEREOF OR OF ANY AIRTIME SERVICES, COST OF SUBSTITUTE GOODS, COSTS OF COVER, FACILITIES OR SERVICES, COST OF CAPITAL, OR OTHER SIMILAR PECUNIARY LOSSES, WHETHER OR NOT SUCH DAMAGES 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
200 Bath Road
Slough, Berkshire SL1 3XE

United Kingdom

Published in Canada