Deprecated and discontinued features and APIs

Consider carefully how removed features and API elements affect your IBM MobileFirst Foundation environment.

Jump to

- Discontinued features and features that are not included in v8.0
- Server-side API Changes
- Client-side API Changes

Discontinued features and features that are not included in v8.0

IBM MobileFirst Foundation v8.0 is radically simplified compared to the previous version. As a result of this simplification, some features that were available in V7.1 are discontinued in v8.0. In most cases, an alternative way to implement the features is suggested. These features are marked discontinued. Some other features that exist in V7.1. are not in v8.0, but not as a consequence of the new design of v8.0. To distinguish these excluded features from the features that are discontinued from v8.0, they are marked not in v8.0.

Feature	Status and replacement path
MobileFirst Studio is replaced by MobileFirst Studio plug-in for Eclipse.	Replaced by MobileFirst Studio plug-in for Eclipse empowered by standard and community-base Eclipse plug-ins. You can develop hybrid applications directly Cordova CLI or with a Cordova enabled IDE such as Visual Studio Code, Eclipse, IntelliJ, and others.For more information about using eclipse as a Cordova e MobileFirst Studio plug-in for managing Cordova projects in Eclipse (file:////home/travis/build/MFPSamples/DevCenter/_site/tutorials/en/foundation/8.0/applica development/using-mobilefirst-cli-in-eclipse/). You can develop adapters with Apache Maven or a maven-enabled IDE such as Eclipse, IntelliJ, and others. For more information about developing adapters category (file:////home/travis/build/MFPSamples/DevCenter/_site/tutorials/en/foundation/8.0/adapters). For more information about using Eclipse as a Maven e Developing Adapters in Eclipse tutorial (file:////home/travis/build/MFPSamples/DevCenter/_site/tutorials/en/foundation/8.0/adapters/developing-adapters/). Install IBM MobileFirst Foundation Developer Kit to test adapters and applications with MobileFirst Development Server. You can also access MobileFirst deve SDKs if you do not want to download them from Internet-based repositories such as NPM, Maven, Cocoapod, or NuGet. For more information about IBM Mob Developer Kit, see The IBM MobileFirst Foundation Developer Kit (file:////home/travis/build/MFPSamples/DevCenter/_site/tutorials/en/foundation/8.0/installatio configuration/development/mobilefirst/).
Skins, Shells, the Setting page, minification, and JavaScript UI elements are discontinued for hybrid applications.	Discontinued. Hybrid applications are developed directly with the Apache Cordova. For more information about replacing skins, shells, the Setting page, and r Removed components and Comparison of Cordova apps developed with v8.0 versus v7.1 and before.
Sencha Touch can no longer be imported into MobileFirst projects for hybrid applications.	Discontinued. MobileFirst hybrid applications are developed directly with the Apache Cordova, and the MobileFirst features are provided as Cordova plug-ins. Touch documentation to integrate Sencha Touch and Cordova.
The encrypted cache is discontinued.	Discontinued. To store encrypted data locally, use JSONStore. For more information about JSONStore, see the JSONStore tutorial (file:////home/travis/build/MFPSamples/DevCenter/_site/tutorials/en/foundation/8.0/application-development/jsonstore).
Triggering Direct Update on demand is not in v8.0. The client application checks for Direct Update when it obtains the OAuth token for a session. You cannot program a client application to check for direct updates at a different point in time in v8.0.	Not in v8.0.

Adapters with session-dependency configuration. In V7.1.0, you can configure MobileFirst Server to work in session-independent mode (default) or in session-dependent mode. Beginning with v8.0, session-dependent mode is no longer supported. The server is inherently independent of the HTTP session, and no related configuration is required.	Discontinued.
Attribute store over IBM WebSphere eXtreme Scale is not supported in v8.0.	Not in v8.0.
Service discovery and adapter generation for IBM Business Process Manager (IBM BPM) process applications, Microsoft Azure Marketplace DataMarket, OData RESTful APIs, RESTful resources, Services that are exposed by an SAP Netweaver Gateway, and Web Services is not in v8.0.	Not in v8.0.
The JMS JavaScript adapter is not in v8.0.	Not in v8.0.
The SAP Gateway JavaScript adapter is not in v8.0.	Not in v8.0.
The SAP JCo JavaScript adapter is not in v8.0.	Not in v8.0.
The Cast Iron JavaScript adapter in not in v8.0.	Not in v8.0.

The OData and Microsoft Azure OData JavaScript adapters are not in v8.0.	Not in v8.0.
Push notification support for USSD is not supported in v8.0.	Discontinued.
Event-based push notifications is not supported in v8.0.	Discontinued. Use the push notification service. For more information on migrating to push notification service, see topic Migrating to push notifications from e notifications.
Security: User-certificate authentication. v8.0 does not include any predefined security check to authenticate users with X.509 client-side certificates.	Not in v8.0.
Security: Integration with IBM Trusteer . v8.0 does not include any predefined security check or challenge to test IBM Trusteer risk factors.	Not in v8.0. Use IBM Trusteer Mobile SDK.
Security: Device provisioning and device auto-provisioning.	Discontinued. Note: Device provisioning is handled in the normal authorization flow. Device data is automatically collected during the registration process of the security flow information about the security flow, see End-to-end authorization flow.
Security: Configuration file for obfuscating Android code with ProGuard. v8.0 does not include the predefined proguard-project.txt configuration file for Android ProGuard obfuscation with a MobileFirst Android application.	Not in v8.0.
Security: Adapter based authentication is replaced. Authentication uses the OAuth protocol and is implemented with security checks.	Replaced by a security check based implementation.

Security: LDAP login. v8.0 does not include any predefined security check to authenticate users with an LDAP server. Instead, for WebSphere Application Server or WebSphere Application Server Liberty use the application server or a gateway to map an Identity Provider such as LDAP to LTPA, and generate an OAuth token for the user by using an LTPA security check.	Not in v8.0. Replaced by an LTPA security check for WebSphere Application Server or WebSphere Application Server Liberty.
Authentication configuration of the HTTP adapter. The predefined HTTP adapter does not support the connection as a user to a remote server.	Not in v8.0. Edit the source code of the HTTP adapter and add the authentication code. Use `MFP.Server.invokeHttp` to add identification tokens to the HTTP request's
Security Analytics, the ability to monitor MobileFirst security framework's events with MobileFirst Analytics Console is not in v8.0.	Not in v.8.0.
The event source-based model for push notifications is discontinued and replaced by the tag-based push service model.	Discontinued and replaced by the tag-based push service model.
Unstructured Supplementary Service Data (USSD) support is not in v8.0.	Not in v8.0.
Cloudant used as a database for MobileFirst Server in not supported in v8.0.	Not in v8.0.

Geolocation: The geolocation support is discontinued in IBM MobileFirst Foundation v8.0. The REST API for beacons and for mediators is discontinued. The client-side and server-side API WL.Geo and WL.Device are discontinued.	Discontinued. Use the native device API or third-party Cordova plug-ins for geolocation.
The MobileFirst Data Proxy feature is discontinued. The Cloudant IMFData and CloudantToolkit APIs are also discontinued.	Discontinued. For more information about replacing the IMFData and CloudantToolkit APIs in your apps, see Migrating apps storing mobile data in Cloudant w Cloudant SDK.
The IBM Tealeaf SDK is no longer bundled with IBM MobileFirst Foundation.	Discontinued. Use IBM Tealeaf SDK. For more information, see Tealeaf installation and implementation in an Android application (https://www.ibm.com/support/knowledgecenter/TLSDK/AndroidGuide1010/CFs/TLAnddLggFrwkInstandImpl/TealeafAndroidLoggingFrameworkInstallationAn cp=SS2MBL_9.0.2%2F5-0-1-0&lang=en) and Tealeaf iOS Logging Framework Installation and Implementation (https://www.ibm.com/support/knowledgecenter/TLSDK/iOSGuide1010/CFs/TLiOSLggFrwkInstandImpl/TealeafIOSLoggingFrameworkInstallationAndImpleme cp=SS2MBL_9.0.2%2F5-0-3-1&lang=en) in the IBM Tealeaf Customer Experience documentation.
IBM MobileFirst Platform Test Workbench is not bundled with IBM MobileFirst Foundation	Discontinued.
BlackBerry, Adobe AIR, Windows Silverlight are not supported by IBM MobileFirst Foundation v8.0. No SDK is provided for these platforms.	Discontinued.

Server-side API Changes

To migrate the server side of your MobileFirst application, take into account the changes to the APIs.
The following tables list the discontinued server-side API elements in v8.0, deprecated server-side API elements in v8.0, and suggested migration paths. For more information about migrating the server side of your application,

JavaScript API elements discontinued in v8.0

Security

API Element	Replacement Path
`WL.Server.getActiveUer`, `WL.Server.getCurrentUserIdentity`, `WL.Server.getCurrentDeviceIdentity`, `WL.Server.setActiveUser`, `WL.Server.getClientId`, `WL.Server.getClientDeviceContext`, `WL.Server.setApplicationContext`	Use `MFP.Server.getAuthenticatedUser` instead.

Event Source

API Element	Replacement Path
`WL.Server.createEventSource`	Use `MFP.Server.getAuthenticatedUser` instead.
`WL.Server.setEventHandlers`	To migrate from Event source-based notifications to tag-based notifications, see Migrating to push notifications from event source-based notifications.
`WL.Server.createEventHandler`	

API Element	Replacement Path
`WL.Server.createSMSEventHandler`	To send SMS messages, use the push service REST API. For more information, see Sending Notifications (../../../notifications/sending-notifications).
`WL.Server.createUSSDEventHandler`	Integrate USSD by using third-party services.

Push

API Element	Replacement Path
`WL.Server.getUserNotificationSubscription`, `WL.Server.notifyAllDevices`, `WL.Server.sendMessage`, `WL.Server.notifyDevice`, `WL.Server.notifyDeviceSubscription`, `WL.Server.notifyAll`, `WL.Server.createDefaultNotification`, `WL.Server.submitNotification`	To migrate from Event source-based notifications to tag-based notifications, see Migrating to push notifications from event source-based notifications.
`WL.Server.subscribeSMS`	Use the REST API Push Device Registration (POST) to register the device. To send and receive SMS notifications, provide the phoneNumber in the payload while invoking the API.
`WL.Server.unsubscribeSMS`	Use the REST API Push Device Registration (DELETE) to unregister the device.
`WL.Server.getSMSSubscription`	Use the REST API Push Device Registration GET) to get the device registrations.

Location Services

API Element	Replacement Path
`WL.Geo.*`	Integrate Location services by using third-party services.

WS-Security

API Element	Replacement Path
`WL.Server.signSoapMessage`	Use the WS-Security capabilities of WebSphere Application Server.

Java API elements discontinued in v8.0

Security

API Element	Replacement Path
`SecurityAPI.getSecurityContext`	Use AdapterSecurityContext instead.

Push

API Element	Replacement Path
`PushAPI.sendMessage(INotification notification, String applicationId)`	To migrate from Event source-based notifications to tag-based notifications, see Migrating to push notifications from event source-based notifications.
`INotification PushAPI.buildNotification();`	To migrate from Event source-based notifications to tag-based notifications, see Migrating to push notifications from event source-based notifications.
`UserSubscription PushAPI.getUserSubscription(String eventSource, String userId)`	To migrate from Event source-based notifications to tag-based notifications, see Migrating to push notifications from event source-based notifications.

Adapters

API Element	Replacement Path
`AdaptersAPI` interface in the `com.worklight.adapters.rest.api package`	Use the `AdaptersAPI` interface in the `com.ibm.mfp.adapter.api` package instead.
`AnalyticsAPI` interface in the `com.worklight.adapters.rest.api` package	Use the `AnalyticsAPI` interface in the `com.ibm.mfp.adapter.api` package instead.
`ConfigurationAPI` interface in the `com.worklight.adapters.rest.api` package	Use the `ConfigurationAPI` interface in the `com.ibm.mfp.adapter.api` package instead.
`OAuthSecurity` annotation in the `com.worklight.core.auth` package	Use the `OAuthSecurity` annotation in the `com.ibm.mfp.adapter.api` package instead.
`MFPJAXRSApplication` class in the `com.worklight.wink.extensions` package	Use the `MFPJAXRSApplication` class in the `com.ibm.mfp.adapter.api` package instead.
`WLServerAPI` interface in the `com.worklight.adapters.rest.api` package	Use the JAX-RS `Context` annotation to access the MobileFirst API interfaces directly.
`WLServerAPIProvider` class in the `com.worklight.adapters.rest.api` package	Use the JAX-RS `Context` annotation to access the MobileFirst API interfaces directly.

Client-side API Changes

The following changes in the APIs are relevant to migrating your MobileFirst client application.
The following tables list the discontinued client-side API elements in V8.0.0, deprecated client-side API elements in V8.0.0, and suggested migration paths.

JavaScript APIs

These JavaScript APIs that affect the user interface are no longer supported in v8.0. They can be replaced with available third-party Cordova plug-ins, or by creating custom Cordova plug-ins.

API Element	Migration Path
`WL.BusyIndicator`, `WL.OptionsMenu`, `WL.TabBar`, `WL.TabBarItem`	Use Cordova plug-ins or HTML 5 elements.
`WL.App.close`	Handle this event outside of MobileFirst.
`WL.App.copyToClipboard()`	Use Cordova plug-ins providing this functionality.
`WL.App.openUrl(url, target, options)`	Use Cordova plug-ins providing this functionality. **Note:** For your information, the Cordova **InAppBrowser** plug-in provides this feature.
`WL.App.overrideBackButton(callback)`, `WL.App.resetBackButton()`	Use Cordova plug-ins providing this functionality. **Note:** For your information, the Cordova **backbutton** plug-in provides this feature.
`WL.App.getDeviceLanguage()`	Use Cordova plug-ins providing this functionality. **Note:** For your information, the Cordova **cordova-plugin-globalization** plug-in provides this feature.
`WL.App.getDeviceLocale()`	Use Cordova plug-ins providing this functionality. **Note:** For your information, the Cordova **cordova-plugin-globalization** plug-in provides this feature.
`WL.App.BackgroundHandler`	To run a custom handler function, use the standard Cordova pause event listener. Use a Cordova plug-in that provides privacy and prevents iOS and Android systems and users from taking snapshots or screen captures. For more information, see the description of the **PrivacyScreenPlugin (https://github.com/devgeeks/PrivacyScreenPlugin)**.
`WL.Client.close`, `WL.Client.restore`, `WL.Client.minimize`	The functions were provided to support the Adobe AIR platform, which is not supported by IBM MobileFirst Platform V8.0.0.
`WL.Toast.show(string)`	Use Cordova plug-ins for Toast.

This set of APIs is no longer supported in v8.0.

API Element	Migration Path
`WL.Client.checkForDirectUpdate(options)`	No replacement. **Note:** You can call `WLAuthorizationManager.obtainAccessToken` to trigger a direct update if one is available. The access to a security token triggers a direct update if one is available on the server. But you cannot trigger Direct Update on demand.
`WL.Client.setSharedToken({key: myName, value: myValue})`, `WL.Client.getSharedToken({key: myName})`, `WL.Client.clearSharedToken({key: myName})`	No replacement.
`WL.Client.isConnected()`, `connectOnStartup` init option	Use `WLAuthorizationManager.obtainAccessToken` to check connectivity to the server and apply application management rules.
`WL.Client.setUserPref(key,value, options)`, `WL.Client.setUserPrefs(userPrefsHash, options)`, `WL.Client.deleteUserPrefs(key, options)`	No replacement. You can use an adapter and the `MFP.Server.getAuthenticatedUser` API to manage user preferences.
`WL.Client.getUserInfo(realm, key)`, `WL.Client.updateUserInfo(options)`	No replacement.
`WL.Client.logActivity(activityType)`	Use `WL.Logger`.
`WL.Client.login(realm, options)`	Use `WLAuthorizationManager.login`. To get started with authentication and security, see the Authentication and Security tutorials.
`WL.Client.logout(realm, options)`	Use `WLAuthorizationManager.logout`.
`WL.Client.obtainAccessToken(scope, onSuccess, onFailure)`	Use `WLAuthorizationManager.obtainAccessToken`.
`WL.Client.transmitEvent(event, immediate)`, `WL.Client.purgeEventTransmissionBuffer()`, `WL.Client.setEventTransmissionPolicy(policy)`	Create a custom adapter for receiving notifications of these events.
`WL.Device.getContext()`, `WL.Device.startAcquisition(policy, triggers, onFailure)`, `WL.Device.stopAcquisition()`, `WL.Device.Wifi`, `WL.Device.Geo.Profiles`, `WL.Geo`	Use native API or third-party Cordova plug-ins for GeoLocation.
`WL.Client.makeRequest (url, options)`	Create a custom adapter that provides the same functionality
`WLDevice.getID(options)`	Use Cordova plug-ins providing this functionality. **Note:** For your information, `device.uuid` from the c**ordova-plugin-device** plug-in provides this feature.
`WL.Device.getFriendlyName()`	Use `WL.Client.getDeviceDisplayName`
`WL.Device.setFriendlyName()`	Use `WL.Client.setDeviceDisplayName`

API Element	Migration Path
`WL.Device.getNetworkInfo(callback)`	Use Cordova plug-ins providing this functionality. **Note:** For your information, the **cordova-plugin-network-information** plug-in provides this feature.
`WLUtils.wlCheckReachability()`	Create a custom adapter to check server availability.
`WL.EncryptedCache`	Use JSONStore to store encrypted data locally. JSONStore is in the **cordova-plugin-mfp-jsonstore** plug-in. For more information, see JSONStore (../../../application-development/jsonstore).
`WL.SecurityUtils.remoteRandomString(bytes)`	Create a custom adapter that provides the same functionality.
`WL.Client.getAppProperty(property)`	You can retrieve the app version property by using the **cordova-plugin-appversion** plug-in. The version that is returned is the native app version (Android and iOS only).
`WL.Client.Push.*`	Use JavaScript client-side push API from the **cordova-plugin-mfp-push** plug-in.
`WL.Client.Push.subscribeSMS(alias, adapterName, eventSource, phoneNumber, options)`	Use `MFPPush.registerDevice(org.json.JSONObject options, MFPPushResponseListener listener)` to register the device for push and SMS.
`WLAuthorizationManager.obtainAuthorizationHeader(scope)`	Use `WLAuthorizationManager.obtainAccessToken` to obtain a token for the required scope.
`WLClient.getLastAccessToken(scope)`	Use `WLAuthorizationManager.obtainAccessToken`
`WLClient.getLoginName()`, `WL.Client.getUserName(realm)`	No replacement
`WL.Client.getRequiredAccessTokenScope(status, header)`	Use `WLAuthorizationManager.isAuthorizationRequired` and `WLAuthorizationManager.getResourceScope`.
`WL.Client.isUserAuthenticated(realm)`	No replacement
`WLUserAuth.deleteCertificate(provisioningEntity)`	No replacement
`WL.Trusteer.getRiskAssessment(onSuccess, onFailure)`	No replacement
`WL.Client.createChallengeHandler(realmName)`	To create a challenge handler for handling custom gateway challenges, use `WL.Client.createGatewayChallengeHandler(gatewayName)`. To create a challenge handler for handling MobileFirst security-check challenges, use `WL.Client.createSecurityCheckChallengeHandler(securityCheckName)`.
`WL.Client.createWLChallengeHandler(realmName)`	Use `WL.Client.createSecurityCheckChallengeHandler(securityCheckName)`.
`challengeHandler.isCustomResponse()` where challengeHandler is a challenge-handler object that is returned by `WL.Client.createChallengeHandler()`	Use `gatewayChallengeHandler.canHandleResponse()` where `gatewayChallengeHandler` is a challenge-handler object that is returned by `WL.Client.createGatewayChallengeHandler()`.
`wlChallengeHandler.processSucccess()` where `wlChallengeHandler` is a challenge-handler object that is returned by `WL.Client.createWLChallengeHandler()`	Use `securityCheckChallengeHandler.handleSuccess()` where `securityCheckChallengeHandler` is a challenge-handler object that is returned by `WL.Client.createSecurityCheckChallengeHandler()`.
`WL.Client.AbstractChallengeHandler.submitAdapterAuthentication()`	Implement similar logic in your challenge handler. For custom gateway challenge handlers, use a challenge-handler object that is returned by `WL.Client.createGatewayChallengeHandler()`. For MobileFirst security-check challenge handlers, use a challenge-handler object that is returned by `WL.Client.createSecurityCheckChallengeHandler()`.
`WL.Client.createProvisioningChallengeHandler()`	No replacement. Device provisioning is now handled automatically by the security framework.

Deprecated JavaScript APIs

API Element	Migration Path
`WLClient.invokeProcedure(WLProcedureInvocationData invocationData,WLResponseListener responseListener)`, `WL.Client.invokeProcedure(invocationData, options)`, `WLClient.invokeProcedure(WLProcedureInvocationData invocationData, WLResponseListener responseListener, WLRequestOptions requestOptions)`, `WLProcedureInvocationResult`	Use the `WLResourceRequest` instead. **Note:** The implementation of `invokeProcedure` uses `WLResourceRequest`.
`WLClient.getEnvironment`	Use Cordova plug-ins providing this functionality. **Note:** For your information, the **device.platform** plug-in provides this feature.
`WLClient.getLanguage`	Use Cordova plug-ins providing this functionality. **Note:** For your information, the **cordova-plugin-globalization** plug-in provides this feature.
`WL.Client.connect(options)`	Use `WLAuthorizationManager.obtainAccessToken` to check connectivity to the server and apply application management rules.

Android APIs

Discontinued Android API elements

API Element	Migration Path
`WLConfig WLClient.getConfig()`	No replacement.
`WLDevice WLClient.getWLDevice()`, `WLClient.transmitEvent(org.json.JSONObject event)`, `WLClient.setEventTransmissionPolicy(WLEventTransmissionPolicy policy)`, `WLClient.purgeEventTransmissionBuffer()`	Use Android API or third-party packages for GeoLocation.
`WL.Client.getUserInfo(realm, key)`, `WL.Client.updateUserInfo(options)`	No replacement.
`WL.Client.getUserInfo(realm, key`, `WL.Client.updateUserInfo(options)`	No replacement.
`WLClient.checkForNotifications()`	Use `WLAuthorizationManager.obtainAccessToken("", listener)` to check connectivity to the server and apply application management rules.
`WLClient.login(java.lang.String realmName, WLRequestListener listener, WLRequestOptions options)`, `WLClient.login(java.lang.String realmName, WLRequestListener listener)`	Use `AuthorizationManager.login()`
`WLClient.logout(java.lang.String realmName, WLRequestListener listener, WLRequestOptions options)`, `WLClient.logout(java.lang.String realmName, WLRequestListener listener)`	Use `AuthorizationManager.logout()`
`WLClient.obtainAccessToken(java.lang.String scope,WLResponseListener responseListener)`	Use `WLAuthorizationManager.obtainAccessToken(String, WLAccessTokenListener)` to check connectivity to the server and apply application management rules
`WLClient.getLastAccessToken()`, `WLClient.getLastAccessToken(java.lang.String scope)`	Use `AuthorizationManager`
`WLClient.getRequiredAccessTokenScope(int status, java.lang.String header)`	Use `AuthorizationManager`
`WLClient.logActivity(java.lang.String activityType)`	Use `com.worklight.common.Logger`. For more information, see Logger SDK.
`WLAuthorizationPersistencePolicy`	No replacement. To implement authorization persistence, store the authorization token in the application code and create custom HTTP requests.
`WLSimpleSharedData.setSharedToken(myName, myValue)`, `WLSimpleSharedData.getSharedToken(myName)`, `WLSimpleSharedData.clearSharedToken(myName)`	Use the Android APIs to share tokens across applications.
`WLUserCertificateManager.deleteCertificate(android.content.Context context)`	No replacement
`BaseChallengeHandler.submitFailure(WLResponse wlResponse)`	Use `BaseChallengeHandler.cancel()`
`ChallengeHandler`	For custom gateway challenges, use `GatewayChallengeHandler`. For MobileFirst security-check challenges, use `SecurityCheckChallengeHandler`.
`WLChallengeHandler`	Use `SecurityCheckChallengeHandler`.
`ChallengeHandler.isCustomResponse()`	se `GatewayChallengeHandler.canHandleResponse()`.
`ChallengeHandler.submitAdapterAuthentication`	Implement similar logic in your challenge handler. For custom gateway challenge handlers, use `GatewayChallengeHandler`.

Deprecated Android APIs

API Element	Migration Path
`WLClient.invokeProcedure(WLProcedureInvocationData invocationData, WLResponseListener responseListener)`	Deprecated. Use `WLResourceRequest`. **Note:** The implementation of `invokeProcedure` uses `WLResourceRequest`.
`WLClient.connect(WLResponseListener responseListener)`, `WLClient.connect(WLResponseListener responseListener,WLRequestOptions options)`	Use `WLAuthorizationManager.obtainAccessToken("", listener)` to check connectivity to the server and apply application management rules.

Android APIs depending on the legacy org.apach.http APIs are no longer supported

API Element	Migration Path
`org.apache.http.Header[]` is now deprecated. Therefore, the following methods are removed:	
`org.apache.http.Header[] WLResourceRequest.getAllHeaders()`	Use instead the new `Map<String, List<String>> WLResourceRequest.getAllHeaders()` API.
`WLResourceRequest.addHeader(org.apache.http.Header header)`	Use instead the new `WLResourceRequest.addHeader(String name, String value)` API.
`org.apache.http.Header[] WLResourceRequest.getHeaders(java.lang.String headerName)`	Use instead the new `List<String> WLResourceRequest.getHeaders(String headerName)` API.

API Element	Migration Path
`org.apache.http.Header WLResourceRequest.getFirstHeader(java.lang.String headerName)`	Use instead the new `WLResourceRequest.getHeaders(String headerName)` API.
`WLResourceRequest.setHeaders(org.apache.http.Header[] headers)`	Instead, use the new `WLResourceRequest.setHeaders(Map<String, List<String>> headerMap)` API.
`WLResourceRequest.setHeader(org.apache.http.Header header)`	Instead, use the new `WLResourceRequest.setHeaders(Map<String, List<String>> headerMap)` API.
`org.apache.http.client.CookieStore WLClient.getCookieStore()`	Replaced with `java.net.CookieStore getCookieStore WLClient.getCookieStore()`
`WLClient.setAllowHTTPClientCircularRedirect(boolean isSet)`	No replacement. MFP Client allows circular redirects.
`WLHttpResponseListener`, `WLResourceRequest.send(java.util.HashMap formParameters,WLHttpResponseListener listener)`, `WLResourceRequest.send(org.json.JSONObject json, WLHttpResponseListener listener)`, `WLResourceRequest.send(byte[] data, WLHttpResponseListener listener)`, `WLResourceRequest.send(java.lang.String requestBody,WLHttpResponseListener listener)`, `WLResourceRequest.send(WLHttpResponseListener listener)`, `WLClient.sendRequest(org.apache.http.client.methods.HttpUriRequest request,WLHttpResponseListener listener)`, `WLClient.sendRequest(org.apache.http.client.methods.HttpUriRequest request, WLResponseListener listener)`	Removed due to deprecated Apache HTTP Client dependencies. Create your own request to have full control over the request and response.

The `com.worklight.androidgap.api` package provides the Android platform functionality for Cordova apps. In MobileFirst, a number of changes were made to accommodate the Cordova integration.

API Element	Migration Path
The Android activity was replaced with the Android context.	
`static WL.createInstance(android.app.Activity activity)`	`static WL.createInstance(android.content.Context context)` creates a shared instance.
`static WL.getInstance()`	`static WL.getInstance()` Gets an instance of the WL class. This method cannot be called before `WL.createInstance(Context)`.

Objective-C APIs

Discontinued iOS Objective C APIs

API Element	Migration Path
`[WLClient getWLDevice][WLClient transmitEvent:]`, `[WLClient setEventTransmissionPolicy]`, `[WLClient purgeEventTransmissionBuffer]`	Geolocation removed. Use native iOS or third-party packages for GeoLocation.
`WL.Client.getUserInfo(realm, key)`, `WL.Client.updateUserInfo(options)`	No replacement.
`WL.Client.deleteUserPref(key, options)`	No replacement. You can use an adapter and the `MFP.Server.getAuthenticatedUser` API to manage user preferences.
`[WLClient getRequiredAccessTokenScopeFromStatus]`	Use `WLAuthorizationManager obtainAccessTokenForScope`.
`[WLClient login:withDelegate:]`	Use `WLAuthorizationManager login`.
`[WLClient logout:withDelegate:]`	Use `WLAuthorizationManager logout`.
`[WLClient lastAccessToken]`, `[WLClient lastAccessTokenForScope:]`	Use `WLAuthorizationManager obtainAccessTokenForScope`.
`[WLClient obtainAccessTokenForScope:withDelegate:]`, `[WLClient getRequiredAccessTokenScopeFromStatus:authenticationHeader:]`	Use `WLAuthorizationManager obtainAccessTokenForScope`.
`[WLClient isSubscribedToAdapter:(NSString *) adaptereventSource:(NSString *) eventSource`	Use Objective-C client-side push API for iOS apps from the IBMMobileFirstPlatformFoundationPush framework
`[WLClient - (int) getEventSourceIDFromUserInfo: (NSDictionary *) userInfo]`	Use Objective-C client-side push API for iOS apps from the IBMMobileFirstPlatformFoundationPush framework.
`[WLClient invokeProcedure: (WLProcedureInvocationData *) ]`	Deprecated. Use `WLResourceRequest` instead.
`WLClient sendUrlRequest:delegate:]`	Use `[WLResourceRequest sendWithDelegate:delegate]` instead.
`[WLClient (void) logActivity:(NSString *) activityType]`	Removed. Use an Objective C logger.

API Element	Migration Path
`[WLSimpleDataSharing setSharedToken: myName value: myValue]`, `[WLSimpleDataSharing getSharedToken: myName]]`, `[WLSimpleDataSharing clearSharedToken: myName]`	Use the OS APIs to share tokens across applications.
`BaseChallengeHandler.submitFailure(WLResponse *)challenge`	Use `BaseChallengeHandler.cancel()`.
`BaseProvisioningChallengeHandler`	No replacement. Device provisioning is now handled automatically by the security framework.
`ChallengeHandler`	For custom gateway challenges, use `GatewayChallengeHandler`. For MobileFirst security-check challenges, use `SecurityCheckChallengeHandler`.
`WLChallengeHandler`	Use `SecurityCheckChallengeHandler`.
`ChallengeHandler.isCustomResponse()`	Use `GatewayChallengeHandler.canHandleResponse()`.
`ChallengeHandler.submitAdapterAuthentication`	Implement similar logic in your challenge handler. For custom gateway challenge handlers, use `GatewayChallengeHandler`. For MobileFirst security-check challenge handlers, use `SecurityCheckChallengeHandler`.

Windows C# APIs

Deprecated Windows C# API elements - Classes

API Element	Migration Path
`ChallengeHandler`	For custom gateway challenges, use `GatewayChallengeHandler`. For MobileFirst security-check challenges, use `SecurityCheckChallengeHandler`.
`ChallengeHandler. isCustomResponse()`	Use `GatewayChallengeHandler.canHandleResponse()`.
`ChallengeHandler.submitAdapterAuthentication`	Implement similar logic in your challenge handler. For custom gateway challenge handlers, use `GatewayChallengeHandler`. For MobileFirst security-check challenge handlers, use `SecurityCheckChallengeHandler`.
`ChallengeHandler.submitFailure(WLResponse wlResponse)`	For custom gateway challenge handlers, use `GatewayChallengeHandler.Shouldcancel()`. For MobileFirst security-check challenge handlers, use `SecurityCheckChallengeHandler.ShouldCancel()`.
`WLAuthorizationManager`	Use `WorklightClient.WorklightAuthorizationManager` instead.
`WLChallengeHandler`	Use `SecurityCheckChallengeHandler`.
`WLChallengeHandler.submitFailure(WLResponse wlResponse)`	Use `SecurityCheckChallengeHandler.ShouldCancel()`.
`WLClient`	Use `WorklightClient` instead.
`WLErrorCode`	Not supported.
`WLFailResponse`	Use `WorklightResponse` instead.
`WLResponse`	Use `WorklightResponse` instead.
`WLProcedureInvocationData`	Use `WorklightProcedureInvocationData` instead.
`WLProcedureInvocationFailResponse`	Not supported.
`WLProcedureInvocationResult`	Not supported.
`WLRequestOptions`	Not supported.
`WLResourceRequest`	Not supported.

Deprecated Windows C# API elements - Interfaces

API Element	Migration Path
`WLHttpResponseListener`	Not supported.
`WLResponseListener`	The response will be available as a `WorklightResponse` object
`WLAuthorizationPersistencePolicy`	Not supported.

Last modified on