

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.	<p>Replaced by MobileFirst Studio plug-in for Eclipse empowered by standard and community-base Eclipse plug-ins. You can develop hybrid applications with 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 IDE, see The IBM MobileFirst Foundation Developer Kit (file:///home/travis/build/MFPSPamples/DevCenter/_site/tutorials/en/foundation/8.0/development/using-mobilefirst-cli-in-eclipse/).</p> <p>You can develop adapters with Apache Maven or a maven-enabled IDE such as Eclipse, IntelliJ, and others. For more information about developing adapters, see Developing Adapters in Eclipse tutorial (file:///home/travis/build/MFPSPamples/DevCenter/_site/tutorials/en/foundation/8.0/adapters/developing-adapters).</p> <p>Install IBM MobileFirst Foundation Developer Kit to test adapters and applications with MobileFirst Development Server. You can also access MobileFirst 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 MobileFirst Foundation Developer Kit, see The IBM MobileFirst Foundation Developer Kit (file:///home/travis/build/MFPSPamples/DevCenter/_site/tutorials/en/foundation/8.0/ins/configuration/development/mobilefirst/).</p>
Skins, Shells, the Setting page, minification, and JavaScript UI elements are discontinued for hybrid applications.	<p>Discontinued. Hybrid applications are developed directly with the Apache Cordova. For more information about replacing skins, shells, the Setting page, minification, and JavaScript UI elements, see Removed components and Comparison of Cordova apps developed with v8.0 versus v7.1 and before.</p>
Sencha Touch can no longer be imported into MobileFirst projects for hybrid applications.	<p>Discontinued. MobileFirst hybrid applications are developed directly with the Apache Cordova, and the MobileFirst features are provided as Cordova plugins. For more information about integrating Sencha Touch and Cordova, see Sencha Touch documentation to integrate Sencha Touch and Cordova.</p>
The encrypted cache is discontinued.	<p>Discontinued. To store encrypted data locally, use JSONStore. For more information about JSONStore, see the JSONStore tutorial (file:///home/travis/build/MFPSPamples/DevCenter/_site/tutorials/en/foundation/8.0/application-development/jsonstore).</p>
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.	<p>Not in v8.0.</p>

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 is 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 .
Security: Anti-cross site request forgery (anti-XSRF) realm (<code>wl_antiXSRFRealm</code>) is not required in V8.0.	In V7.1.0, the authentication context is stored in the HTTP session and identified by a session cookie, which is sent by the browser in cross-site request this version is used to protect the cookie transmission against XSRF attacks by using an additional header that is sent from the client to the server. In V8.0.0, the security context is no longer associated with an HTTP session and is not identified by a session cookie. Instead, the authorization is done access token that is passed in the Authorization header. Because the Authorization header is not sent by the browser in cross-site requests, there is no XSRF attacks.

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 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 <code>MFP.Server.invokeHttp</code> to add identification tokens to the HTTP request.
Security Analytics, the ability to monitor MobileFirst security framework's events with MobileFirst Analytics Console is not in v8.0.	Not in v8.0.

Push

API Element

WL.Server.getUserNotificationSubscription, WL.Server.notifyAllDevices, WL.Server.sendMessage, WL.Server.notifyDevice, WL.Server.notifyDeviceSubscription, WL.Server.notifyAll, WL.Server.createDefaultNotification, WL.Server.submitNotification

WL.Server.subscribeSMS

WL.Server.unsubscribeSMS

WL.Server.getSMSSubscription

Location Services

API ElementReplacement Path

WL.Geo.* Integrate Location services by using third-party services.

WS-Security

API Element Replacement Path

WL.Server.signSoapMessageUse the WS-Security capabilities of WebSphere Application Server.

Java API elements discontinued in v8.0

Security

API Element Replacement Path

SecurityAPI.getSecurityContextUse AdapterSecurityContext instead.

Push

API Element

PushAPI.sendMessage(INotification notification, String applicationId)

INotification PushAPI.buildNotification();

UserSubscription PushAPI.getUserSubscription(String eventSource, String userId)

Replacement Path

To migrate from Event source-based notifications to tag-based notifications, see Migrating to push notifications from event source-based notifications.

To migrate from Event source-based notifications to tag-based notifications, see Migrating to push notifications from event source-based notifications.

To migrate from Event source-based notifications to tag-based notifications, see Migrating to push notifications from event source-based notifications.

Adapters

API Element

AdaptersAPI interface in the com.worklight.adapters.rest.api package

AnalyticsAPI interface in the com.worklight.adapters.rest.api package

ConfigurationAPI interface in the com.worklight.adapters.rest.api package

OAuthSecurity annotation in the com.worklight.core.auth package

MFPJAXRSApplication class in the com.worklight.wink.extensions package

WLServerAPI interface in the com.worklight.adapters.rest.api package

WLServerAPIProvider class in the com.worklight.adapters.rest.api package

Replacement Path

Use the AdaptersAPI interface in the com.ibm.mfp.adapter.api package instead.

Use the AnalyticsAPI interface in the com.ibm.mfp.adapter.api package instead.

Use the ConfigurationAPI interface in the com.ibm.mfp.adapter.api package instead.

Use the OAuthSecurity annotation in the com.ibm.mfp.adapter.api package instead.

Use the MFPJAXRSApplication class in the com.ibm.mfp.adapter.api package instead.

Use the JAX-RS Context annotation to access the MobileFirst API interfaces directly.

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

WL.BusyIndicator, WL.OptionsMenu,

WL.TabBar, WL.TabBarItem

WL.App.close

WL.App.copyToClipboard()

WL.App.openUrl(url, target, options)

WL.App.overrideBackButton(callback),

WL.App.resetBackButton()

WL.App.getDeviceLanguage()

WL.App.getDeviceLocale()

WL.App.BackgroundHandler

WL.Client.close, WL.Client.restore,

WL.Client.minimize

WL.Toast.show(string)

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

Migration Path

Use Cordova plug-ins or HTML 5 elements.

Handle this event outside of MobileFirst.

Use Cordova plug-ins providing this functionality.

Use Cordova plug-ins providing this functionality. **Note:** For your information, the Cordova **InAppBrowser** plug-in provides this feature.

Use Cordova plug-ins providing this functionality. **Note:** For your information, the Cordova **backbutton** plug-in provides this feature.

Use Cordova plug-ins providing this functionality. **Note:** For your information, the Cordova **cordova-plugin-globalization** plug-in provides this feature.

Use Cordova plug-ins providing this functionality. **Note:** For your information, the Cordova **cordova-plugin-globalization** plug-in provides this feature.

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>).

The functions were provided to support the Adobe AIR platform, which is not supported by IBM MobileFirst Platform V8.0.0.

Use Cordova plug-ins for Toast.

API Element

WL.Client.checkForDirectUpdate(options)

Migration Path

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.

API Element

```
WL.Client.setSharedToken({key: myName, value: myValue}),  
WL.Client.getSharedToken({key: myName}),  
WL.Client.clearSharedToken({key: myName})
```

```
WL.Client.isConnected(), connectOnStartup init option
```

```
WL.Client.setUserPref(key,value, options),  
WL.Client.setUserPrefs(userPrefsHash, options),  
WL.Client.deleteUserPrefs(key, options)  
WL.Client.getUserInfo(realm, key),  
WL.Client.updateUserInfo(options)  
WL.Client.logActivity(activityType)
```

```
WL.Client.login(realm, options)
```

```
WL.Client.logout(realm, options)  
WL.Client.obtainAccessToken(scope, onSuccess, onFailure)  
WL.Client.transmitEvent(event, immediate),  
WL.Client.purgeEventTransmissionBuffer(),  
WL.Client.setEventTransmissionPolicy(policy)  
WL.Device.getContext(), WL.Device.startAcquisition(policy,  
triggers, onFailure), WL.Device.stopAcquisition(), WL.Device.Wifi,  
WL.Device.Geo.Profiles, WL.Geo  
WL.Client.makeRequest (url, options)
```

```
WLDevice.getID(options)
```

```
WL.Device.getFriendlyName()  
WL.Device.setFriendlyName()
```

```
WL.Device.getNetworkInfo(callback)
```

```
WLUtils.wlCheckReachability()
```

```
WL.EncryptedCache
```

```
WL.SecurityUtils.remoteRandomString(bytes)
```

```
WL.Client.getAppProperty(property)
```

```
WL.Client.Push.*  
WL.Client.Push.subscribeSMS(alias, adapterName, eventSource,  
phoneNumber, options)
```

```
WLAuthorizationManager.obtainAuthorizationHeader(scope)
```

```
WLClient.getLastAccessToken(scope)  
WLClient.getLoginName(), WL.Client.getUserName(realm)
```

```
WL.Client.getRequiredAccessTokenScope(status, header)
```

```
WL.Client.isUserAuthenticated(realm)  
WLUserAuth.deleteCertificate(provisioningEntity)  
WL.Trusteer.getRiskAssessment(onSuccess, onFailure)
```

```
WL.Client.createChallengeHandler(realmName)
```

```
WL.Client.createWLChallengeHandler(realmName)  
challengeHandler.isCustomResponse() where challengeHandler is a  
challenge-handler object that is returned by  
WL.Client.createChallengeHandler()  
wlChallengeHandler.processSuccess() where wlChallengeHandler is a  
challenge-handler object that is returned by  
WL.Client.createWLChallengeHandler()
```

```
WL.Client.AbstractChallengeHandler.submitAdapterAuthentication()
```

```
WL.Client.createProvisioningChallengeHandler()
```

Deprecated JavaScript APIs

API Element

```
WLClient.invokeProcedure(WLProcedureInvocationData invocationData,WLResponseListener  
responseListener), WL.Client.invokeProcedure(invocationData, options),  
WLClient.invokeProcedure(WLProcedureInvocationData invocationData, WLResponseListener  
responseListener, WLRequestOptions requestOptions), WLProcedureInvocationResult
```

```
WLClient.getEnvironment
```

```
WLClient.getLanguage
```

```
WL.Client.connect(options)
```

Android APIs

Migration Path

No replacement.

Use `WLAuthorizationManager.obtainAccessToken` to check connectivity to the server and apply application management rules.

No replacement. You can use an adapter and the `MFP.Server.getAuthenticatedUser` API to manage user preferences.

No replacement.

Use `WL.Logger`.

Use `WLAuthorizationManager.login`. To get started with authentication and security, see the Authentication and Security tutorials.

Use `WLAuthorizationManager.logout`.

Use `WLAuthorizationManager.obtainAccessToken`.

Create a custom adapter for receiving notifications of these events.

Create a custom adapter that provides the same functionality

Use Cordova plug-ins providing this functionality. **Note:** For your information, `device.uuid` from the **cordova-plugin-device** plug-in provides this feature.

Use `WL.Client.getDeviceDisplayName`

Use `WL.Client.setDeviceDisplayName`

Use Cordova plug-ins providing this functionality. **Note:** For your information, the **cordova-plugin-network-information** plug-in provides this feature.

Create a custom adapter to check server availability.

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`).

Create a custom adapter that provides the same functionality.

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).

Use JavaScript client-side push API from the **cordova-plugin-mfp-push** plug-in.

Use `MFP.Push.registerDevice(org.json.JSONObject options, MFP.Push.ResponseListener listener)` to register the device for push and SMS.

Use `WLAuthorizationManager.obtainAccessToken` to obtain a token for the required scope.

Use `WLAuthorizationManager.obtainAccessToken`

No replacement

Use `WLAuthorizationManager.isAuthorizationRequired` and `WLAuthorizationManager.getResourceScope`.

No replacement

No replacement

No replacement

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)`.

Use `WL.Client.createSecurityCheckChallengeHandler(securityCheckName)`.

Use `gatewayChallengeHandler.canHandleResponse()` where `gatewayChallengeHandler` is a challenge-handler object that is returned by

`WL.Client.createGatewayChallengeHandler()`.

Use `securityCheckChallengeHandler.handleSuccess()` where

`securityCheckChallengeHandler` is a challenge-handler object that is returned by

`WL.Client.createSecurityCheckChallengeHandler()`.

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()`.

No replacement. Device provisioning is now handled automatically by the security framework.

Migration Path

Use the `WLResourceRequest` instead. **Note:** The implementation of `invokeProcedure` uses `WLResourceRequest`.

Use Cordova plug-ins providing this functionality. **Note:** For your information, the **device.platform** plug-in provides this feature.

Use Cordova plug-ins providing this functionality.

Note: For your information, the **cordova-plugin-globalization** plug-in provides this feature.

Use

`WLAuthorizationManager.obtainAccessToken` to check connectivity to the server and apply application management rules.

Discontinued Android API elements

API Element

```
WLCConfig WLCClient.getConfig()  
WLDevice WLCClient.getWLDevice(), WLCClient.transmitEvent(org.json.JSONObject  
event), WLCClient.setEventTransmissionPolicy(WLEventTransmissionPolicy policy),  
WLCClient.purgeEventTransmissionBuffer()  
WLClient.getUserInfo(realm, key), WLClient.updateUserInfo(options)  
WLClient.getUserInfo(realm, key, WLClient.updateUserInfo(options))
```

```
WLCClient.checkForNotifications()
```

```
WLCClient.login(java.lang.String realmName, WLRequestListener listener,  
WLRequestOptions options), WLCClient.login(java.lang.String realmName,  
WLRequestListener listener)  
WLCClient.logout(java.lang.String realmName, WLRequestListener listener,  
WLRequestOptions options), WLCClient.logout(java.lang.String realmName,  
WLRequestListener listener)
```

```
WLCClient.obtainAccessToken(java.lang.String scope, WLResponseListener  
responseListener)
```

```
WLCClient.getLastAccessToken(), WLCClient.getLastAccessToken(java.lang.String  
scope)
```

```
WLCClient.getRequiredAccessTokenScope(int status, java.lang.String header)
```

```
WLCClient.logActivity(java.lang.String activityType)
```

```
WLAuthorizationPersistencePolicy
```

```
WLSimpleSharedData.setSharedToken(myName, myValue),  
WLSimpleSharedData.getSharedToken(myName),  
WLSimpleSharedData.clearSharedToken(myName)  
WLUserCertificateManager.deleteCertificate(android.content.Context context)  
BaseChallengeHandler.submitFailure(WLResponse wlResponse)
```

```
ChallengeHandler
```

```
WLChallengeHandler
```

```
ChallengeHandler.isCustomResponse()
```

```
ChallengeHandler.submitAdapterAuthentication
```

Deprecated Android APIs

API Element

```
WLCClient.invokeProcedure(WLProcedureInvocationData invocationData,  
WLResponseListener responseListener)  
WLCClient.connect(WLResponseListener responseListener),  
WLCClient.connect(WLResponseListener responseListener, WLRequestOptions  
options)
```

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

API Element

org.apache.http.Header[] is now deprecated. Therefore, the following methods are removed:

```
org.apache.http.Header[] WLResourceRequest.getAllHeaders()
```

```
WLResourceRequest.addHeader(org.apache.http.Header header)
```

```
org.apache.http.Header[] WLResourceRequest.getHeaders(java.lang.String headerName)
```

```
org.apache.http.Header WLResourceRequest.getFirstHeader(java.lang.String headerName)
```

```
WLResourceRequest.setHeaders(org.apache.http.Header[] headers)
```

```
WLResourceRequest.setHeader(org.apache.http.Header header)
```

```
org.apache.http.cookie.CookieStore WLCClient.getCookieStore()
```

```
WLCClient.setAllowHTTPClientCircularRedirect(boolean isSet)
```

```
WLHttpResponseListener, WLResourceRequest.send(java.util.HashMap  
formParameters, WLHttpResponseListener listener), WLResourceRequest.send(org.json.JSONObject json,  
WLHttpRequestResponseListener listener), WLResourceRequest.send(byte[] data, WLHttpRequestResponseListener  
listener), WLResourceRequest.send(java.lang.String requestBody, WLHttpRequestResponseListener listener),  
WLResourceRequest.send(WLHttpRequestResponseListener listener),  
WLCClient.sendRequest(org.apache.http.client.methods.HttpUriRequest request, WLHttpRequestResponseListener  
listener), WLCClient.sendRequest(org.apache.http.client.methods.HttpUriRequest request,  
WLResponseListener listener)
```

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.

Migration Path

No replacement.

Use Android API or third-party packages for GeoLocation.

No replacement.

No replacement.

Use WLAAuthorizationManager.obtainAccessToken("", listener) to check connectivity to the server and apply application management rules.

Use AuthorizationManager.login()

Use AuthorizationManager.logout()

Use WLAAuthorizationManager.obtainAccessToken(String, WLAAccessTokenListener) to check connectivity to the server and apply application management rules

Use AuthorizationManager

Use AuthorizationManager

Use com.worklight.common.Logger. For more information, see Logger SDK.

No replacement. To implement authorization persistence, store the authorization token in the application code and create custom HTTP requests.

Use the Android APIs to share tokens across applications.

No replacement

Use BaseChallengeHandler.cancel()

For custom gateway challenges, use GatewayChallengeHandler. For MobileFirst security-check challenges, use SecurityCheckChallengeHandler.

Use SecurityCheckChallengeHandler.

se GatewayChallengeHandler.canHandleResponse().

Implement similar logic in your challenge handler. For custom gateway challenge handlers, use GatewayChallengeHandler.

Migration Path

Deprecated. Use WLResourceRequest. **Note:** The implementation of invokeProcedure uses WLResourceRequest.

Use WLAAuthorizationManager.obtainAccessToken("", listener) to check connectivity to the server and apply application management rules.

Migration Path

Use instead the new Map<String, List<String>>

WLResourceRequest.getAllHeaders() API.

Use instead the new

WLResourceRequest.addHeader(String name, String value) API.

Use instead the new List<String>

WLResourceRequest.getHeaders(String headerName) API.

Use instead the new

WLResourceRequest.getHeaders(String headerName) API.

Instead, use the new

WLResourceRequest.setHeaders(Map<String, List<String>> headerMap) API.

Instead, use the new

WLResourceRequest.setHeaders(Map<String, List<String>> headerMap) API.

Replaced with java.net.CookieStore

getCookieStore WLCClient.getCookieStore()

No replacement. MFP Client allows circular redirects.

Removed due to deprecated Apache HTTP Client dependencies. Create your own request to have full control over the request and response.

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

Objective-C APIs

Discontinued iOS Objective C APIs

API Element	Migration Path
<code>[WLClient getWLDevice]</code> , <code>[WLClient transmitEvent:]</code> , <code>[WLClient setEventTransmissionPolicy]</code> , <code>[WLClient purgeEventTransmissionBuffer]</code> , <code>WL.Client.getUserInfo(realm, key)</code> , <code>WL.Client.updateUserInfo(options)</code>	Geolocation removed. Use native iOS or third-party packages for <code>GeoLocation</code> .
<code>WL.Client.deleteUserPref(key, options)</code>	No replacement.
<code>[WLClient getRequiredAccessTokenScopeFromStatus]</code> , <code>[WLClient login:withDelegate:]</code> , <code>[WLClient logout:withDelegate:]</code> , <code>[WLClient lastAccessToken]</code> , <code>[WLClient lastAccessTokenForScope:]</code> , <code>[WLClient obtainAccessTokenForScope:withDelegate:]</code> , <code>[WLClient getRequiredAccessTokenScopeFromStatus:authenticationHeader:]</code> , <code>[WLClient isSubscribedToAdapter:(NSString *) adapter:reventSource:(NSString *) eventSource</code> <code>[WLClient - (int) getSourceIDFromUserInfo:(NSDictionary *) userInfo]</code> <code>[WLClient invokeProcedure:(WLProcedureInvocationData *)]</code> <code>WLClient sendURLRequest:delegate:]</code> <code>[WLClient (void) logActivity:(NSString *) activityType]</code> <code>[WLSimpleDataSharing setSharedToken: myName value: myValue]</code> , <code>[WLSimpleDataSharing getSharedToken: myName]]</code> , <code>[WLSimpleDataSharing clearSharedToken: myName]</code> <code>BaseChallengeHandler.submitFailure(WLResponse *)challenge</code> <code>BaseProvisioningChallengeHandler</code>	No replacement. You can use an adapter and the <code>MFP.Server.getAuthenticatedUser</code> API to manage user preferences. Use <code>WLAuthorizationManager obtainAccessTokenForScope</code> . Use <code>WLAuthorizationManager login</code> . Use <code>WLAuthorizationManager logout</code> . Use <code>WLAuthorizationManager obtainAccessTokenForScope</code> . Use <code>WLAuthorizationManager obtainAccessTokenForScope</code> . Use Objective-C client-side push API for iOS apps from the <code>IBMMobileFirstPlatformFoundationPush</code> framework Use Objective-C client-side push API for iOS apps from the <code>IBMMobileFirstPlatformFoundationPush</code> framework. Deprecated. Use <code>WLResourceRequest</code> instead. Use <code>[WLResourceRequest sendWithDelegate:delegate]</code> instead. Removed. Use an Objective C logger.
<code>ChallengeHandler</code>	Use the OS APIs to share tokens across applications.
<code>WLChallengeHandler</code> <code>ChallengeHandler.isCustomResponse()</code>	Use <code>BaseChallengeHandler.cancel()</code> . No replacement. Device provisioning is now handled automatically by the security framework. For custom gateway challenges, use <code>GatewayChallengeHandler</code> . For MobileFirst security-check challenges, use <code>SecurityCheckChallengeHandler</code> . Use <code>SecurityCheckChallengeHandler</code> . Use <code>GatewayChallengeHandler.canHandleResponse()</code> . Implement similar logic in your challenge handler. For custom gateway challenge handlers, use <code>GatewayChallengeHandler</code> . For MobileFirst security-check challenge handlers, use <code>SecurityCheckChallengeHandler</code> .
<code>ChallengeHandler.submitAdapterAuthentication</code>	

Windows C# APIs

Deprecated Windows C# API elements - Classes

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

Deprecated Windows C# API elements - Interfaces

API Element	Migration Path
<code>WLHttpRequestListener</code>	Not supported.
<code>WLResponseListener</code>	The response will be available as a <code>WorklightResponse</code> object
<code>WLAuthorizationPersistencePolicy</code>	Not supported.