# Using Direct Update to quickly update your application

## About Direct Update

With Direct Update, hybrid applications (for the Android, iOS, and Windows Phone 8 environments) and Cordova-based applications can be updated "over-the-air" with updated/refreshed versions of the web resources.

This tutorial covers the following topics:

- Benefits and restrictions
- Under the hood
- Internal function - How Direct Update works
- User experience
- Distribution - Working with Direct Update in the field
- Disabling old application versions
- Direct Update authenticity
- Sample application

## Benefits and restrictions

### Benefits

- Using Direct Update, organizations can ensure that users always use the latest version of the application.
- Application versions are better controlled. Users are notified of pending updates or prevented from using obsolete versions.
- Updates that are deployed to the MobileFirst Server are automatically pushed to user devices.
- Updates can be pushed silently without user interaction. This benefit requires that you create a custom Direct Update (see Customizing the UI).

### Restrictions

- The update is for the app web resources only.
- To update native resources, a new app version must be uploaded to the respective app store.
- Android: no restrictions.
- Windows Phone 8: no restrictions.
- iOS:
    - B2C: according to the terms of service of your company; usually at least bug fixes are allowed.
    - B2E: through the iOS Developer Enterprise Program

## Under the hood

Direct Update is based on the MobileFirst Authentication Framework. It comes with a predefined authentication realm and challenge handler, and can be adjusted in the following ways:

1. The security test can be disabled, and instead be explicitly called using the `WL.Client.checkForDirectUpdate` API method
2. Change the `wl.realm.expiration.directUpdate` value in the `worklight.properties` file to a lower value. **Note:** The lower the value, the more network traffic will incur. The value should be configured with care.

   > For more information on configuring the Direct Update review the "Configuring login modules" user documentation topic.

The UI & UX of the Direct Update process can be fully customized by using simple interfaces.

### Server-side customization

By default, the `mobileSecurityTest` has Direct Update enabled in `perSession` mode:

```
<mobileSecurityTest name="mobileTests">
   <testAppAuthenticity/>
   <testDeviceId provisioningType="none" /
>
   <testUser realm="myMobileLoginForm" /
>
   <testDirectUpdate mode="perSession" />
</mobileSecurityTest>
```

To disable Direct Update, change its mode to `disabled`, and similarly to override it to other modes.

```
<testDirectUpdate mode="disabled" />
```

Also by default, the `customSecurityTest` does not have any realms.
To add a Direct Update realm to a custom security test, add a `test` element with the realm name `wl_directUpdateRealm` and define the required mode property:

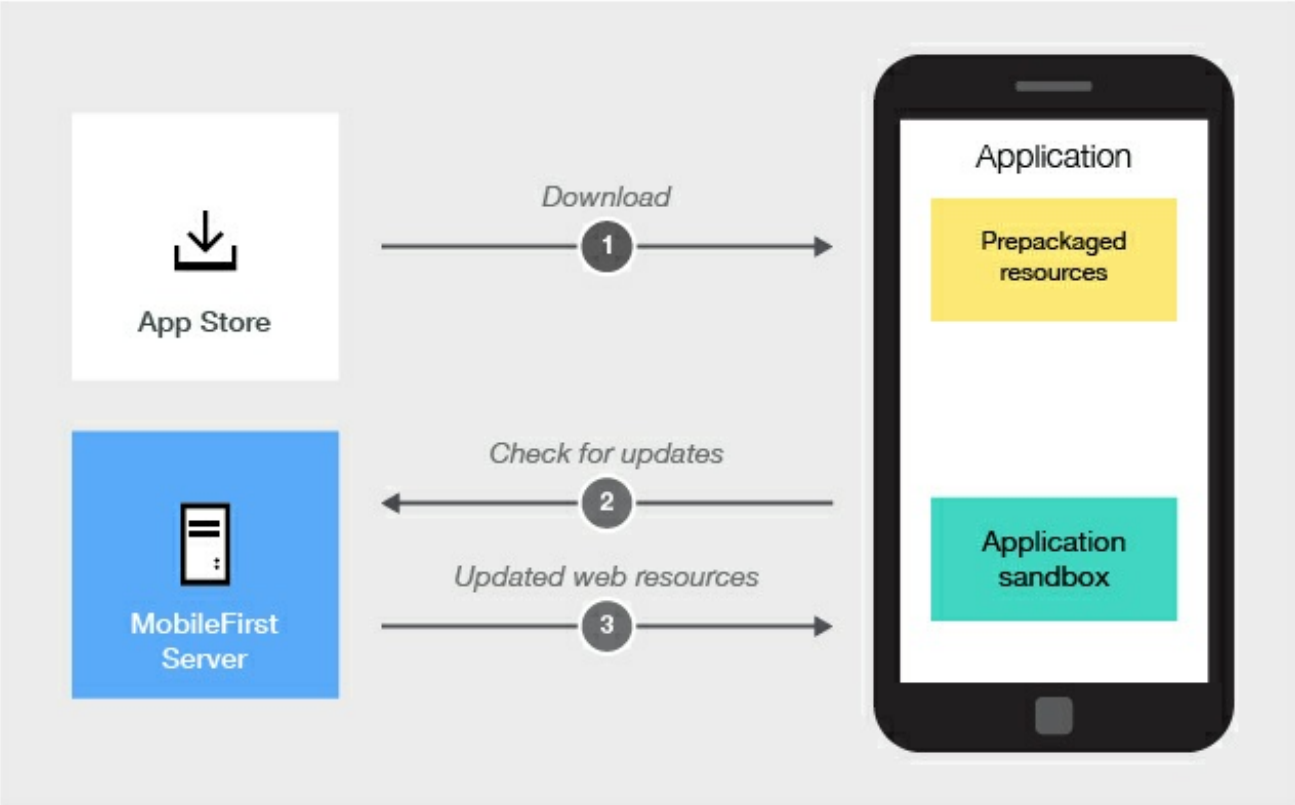
```
<customSecurityTest name="customTests">
   <test realm="wl_directUpdateRealm" mode="perRequest"/
>
</customSecurityTest>
```

> For more information, see the topic about Direct Update as a security realm, in the user documentation.

## Internal function - How Direct Update works

The application web resources are initially packaged with the application to ensure first offline availability. Afterwards, the application checks for updates based on available configurations (per session or per request, as explained above).

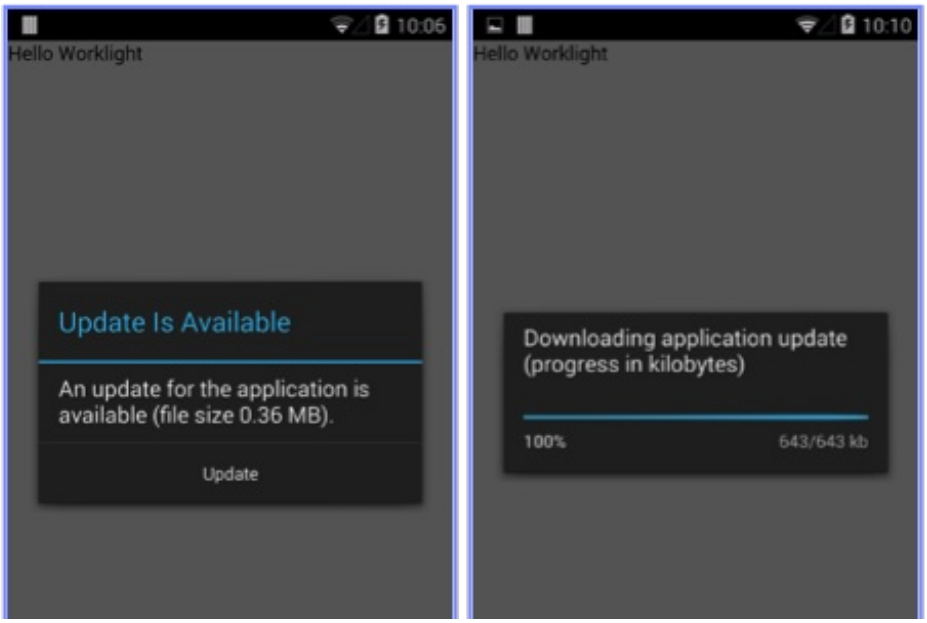
The updated web resources are downloaded when necessary.
After a Direct Update, the application no longer uses prepackaged resources but those resources in the application's sandbox.

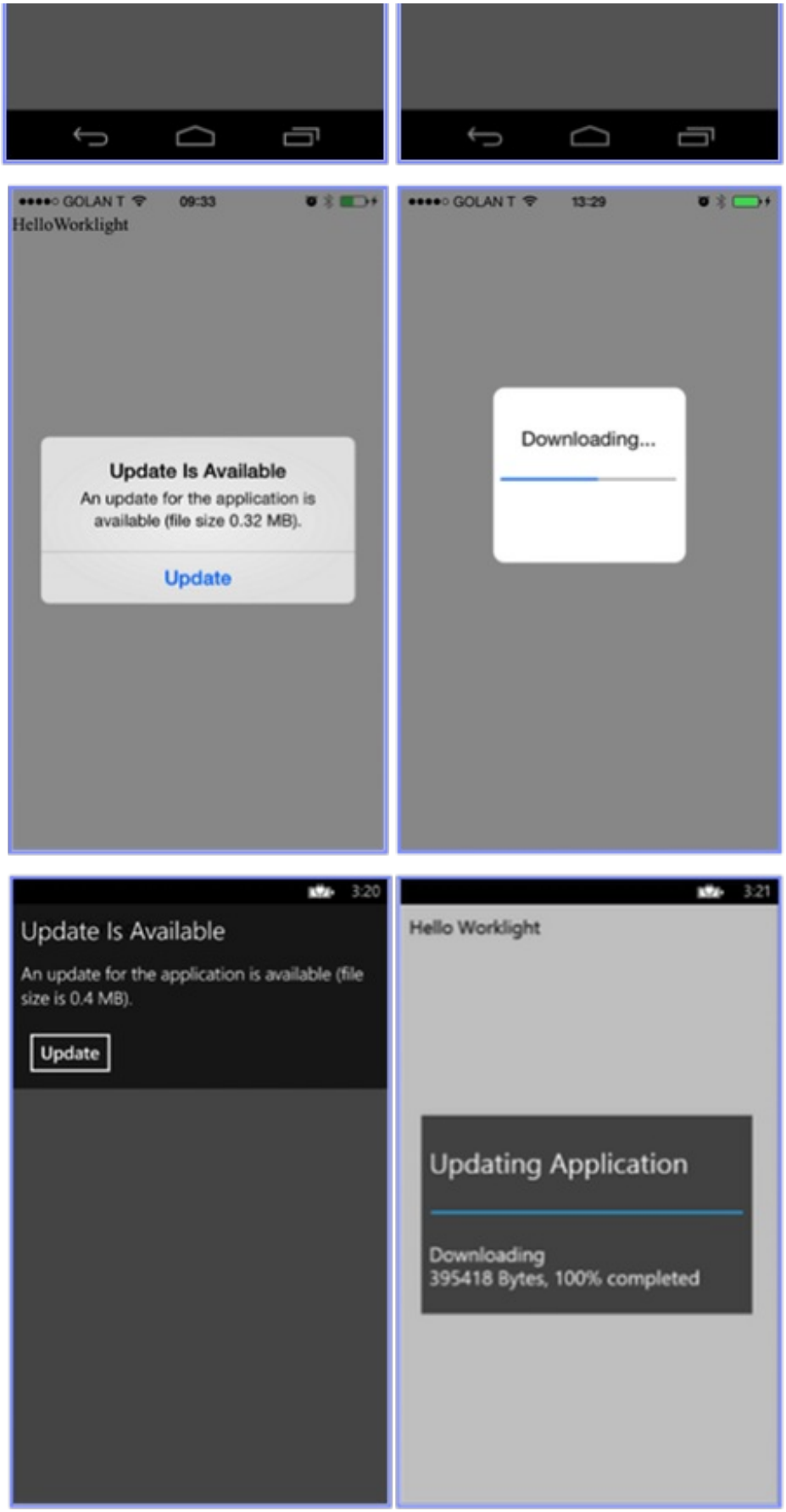



# User experience

## Default behavior

1. By default, after a Direct Update is received, a dialog is displayed and the user is asked whether to begin the update process.
2. After the user approves, a progress bar dialog is displayed and web resources are downloaded.
3. The application is automatically reloaded after the update is complete.






## Differential direct update

With this feature, it is no longer necessary for the client application to download the entire web resources on every update. Instead, only the resources that were changed are downloaded and updated. This reduces download time, conserves bandwidth, and improves overall user experience.

**Important:** A differential update is possible only if the client application's web resources are one version behind the application that is currently deployed on the server. Client applications that are more than one version behind the current deployed application (the application was deployed to the server at least twice since the client application was updated), receive a full update, meaning that the entire web resources are downloaded and updated.

There is no change in the behaviour of applications that were built with previous versions of IBM MobileFirst Platform Foundation.

## Customizing the UI

It is possible to override the default UI and/or UX and create a custom Direct Update behavior altogether.

To do so, override the `handleDirectUpdate` function:

```
wl_DirectUpdateChallengeHandler.handleDirectUpdate = function(directUpdateData, directUpdateConte
xt) {
    // custom Direct Update logic
};
```

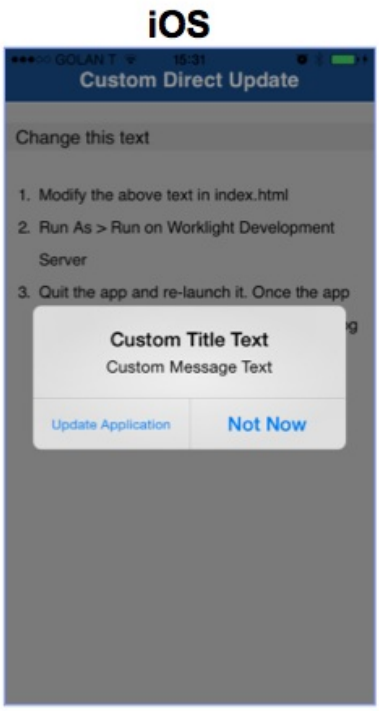
- `directUpdateData` - A JSON object containing the `downloadSize` property that represents the file size (in bytes) of the update package to be downloaded from MobileFirst Server.
- `directUpdateContext` - A JavaScript object exposing the `.start()` and `.stop()` functions, which start and stop the Direct Update flow.

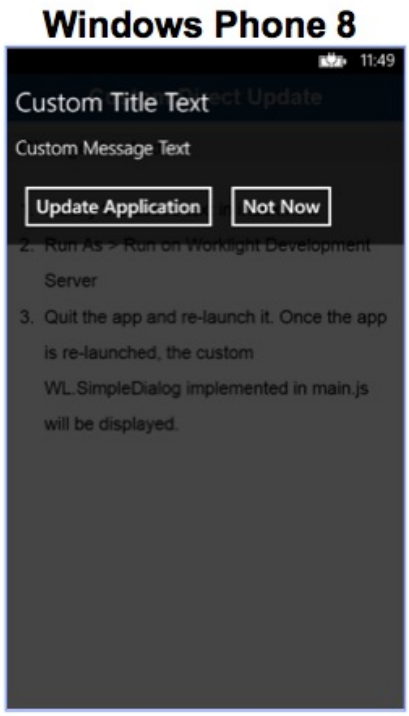
### Example

In the example code below, a custom Direct Update dialog is presented for the user to either continue with the update process or dismiss it.
Examples of a new customized Direct Update UI:

- A WL.SimpleDialog
- A dialog that is created by using a third-party JavaScript framework (such as Dojo or jQuery Mobile)
- Fully native UI by executing a Cordova plug-in
- An alternate HTML file that is presented to the user with options
- And so on…

```
wl_directUpdateChallengeHandler.handleDirectUpdate = function(directUpdateData,
directUpdateContext) {
    // custom WL.SimpleDialog for Direct Update
    var customDialogTitle = 'Custom Title Text';
    var customDialogMessage = 'Custom Message Text';
    var customButtonText1 = 'Update Application';
    var customButtonText2 = 'Not Now';
    WL.SimpleDialog.show(customDialogTitle, customDialogMessage,
        [{
            text : customButtonText1,
            handler : function() {
                directUpdateContext.start();
            }
        },
        {
            text : customButtonText2,
            handler : function() {
                wl_directUpdateChallengeHandler.submitFailure();
            }
        }]
    );
};
```



In the example above, the `submitFailure` API is used to dismiss the Direct Update:

```
wl_directUpdateChallengeHandler.submitFailure();
```

As mentioned, when the developer creates a customized Direct Update experience, the responsibility for its flow now belongs to the developer.

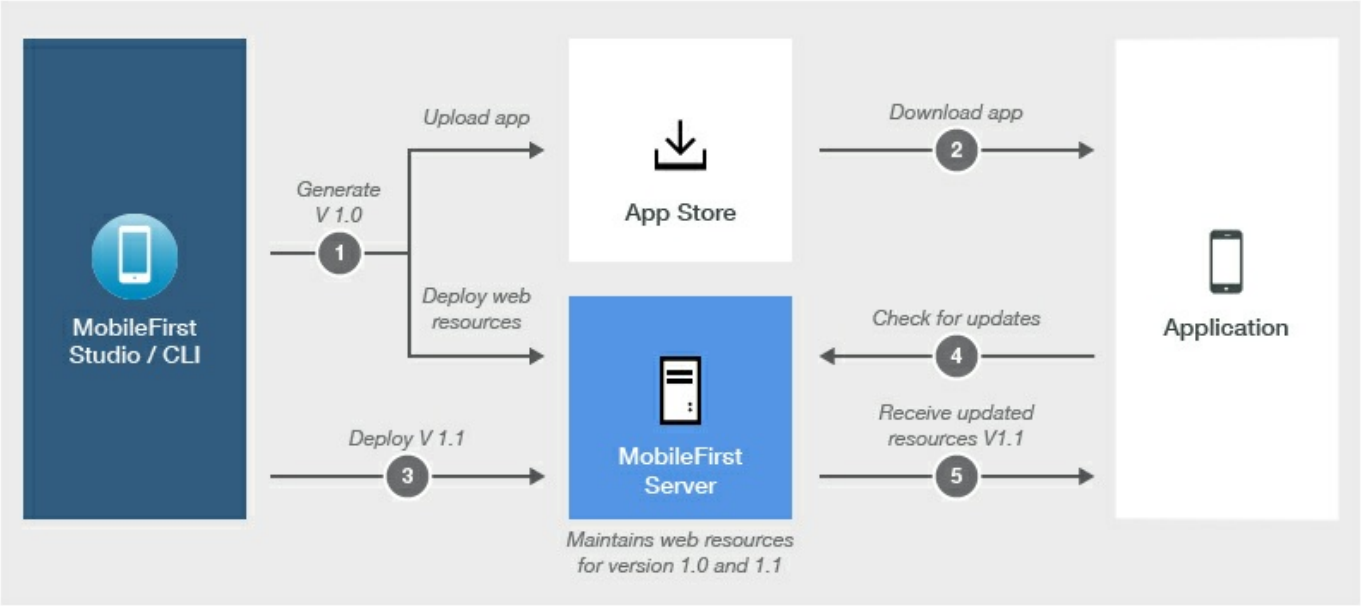
As such, it is important to call `submitFailure()` to notify the MobileFirst framework that the process completed with a "failure". The MobileFirst framework in turn invokes the `onFailure` callback of the invocation that triggered the Direct Update.

Because the update process did not take place, it will occur again the next time it is triggered.

Optionally, a developer can also supply a Direct Update listener to fully control a Direct Update lifecycle. For more information, see the topic about customizing the direct update interface, in the user documentation.

```
directUpdateContext.start(directUpdateCustomListener);
```

## Distribution - Working with Direct Update in the field




(*) During development cycles, testers automatically get recent web resources through internal distribution mechanisms and not through application stores.

## Disabling old application versions

From the MobileFirst Operations Console, it is possible to prevent users from using obsolete versions and to notify users about available updates.

**Clarification:** The Remote Disable feature only prevents users from interacting with MobileFirst Server; that is, it prevents the app from connecting to the server. The application itself is still accessible. Any action in the application that requires server connectivity is blocked.

## Direct Update authenticity

Enabling Direct Update authenticity prevents a 3rd-party attacker from altering the transmitted web resources from the server to the client application (that is, when it is cached in a content delivery network (CDN)).

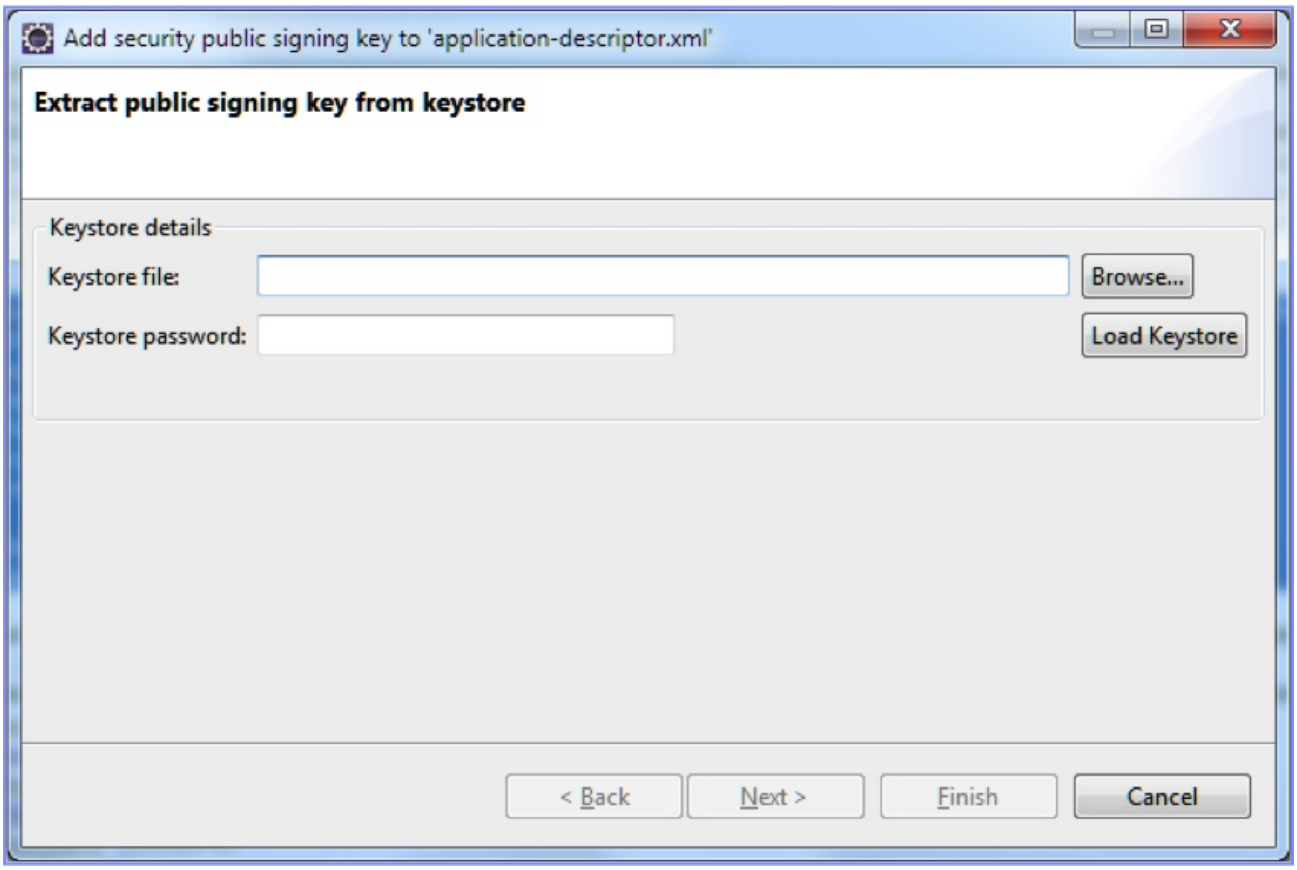
To enable Direct Update authenticity:

1. Generate a certificate and place it in the MobileFirst project in the `server\conf` folder.
2. Edit the MobileFirst Default Certificate section in `server\conf\worklight.properties` with the

certificate keystore information, for example:

```
wl.ca.keystore.path=conf/myCert.jks
wl.ca.keystore.type=jks
wl.ca.keystore.password=myStrongPassword
wl.ca.key.alias=certAlias
wl.ca.key.alias.password=myCertPassword
```

3. Add the certificate public key using the Application Descriptor Design view. To do so:
    1. Right-click the application folder and select **Extract Public Signing Key**.
    2. Follow the on-screen instructions.




The public key can then be found in the Application Descriptor Design view:

The public key can also be found in the Application Descriptor Editor view:

```
<application>
  …
  …
  …
  <directUpdateAuthenticityPublicKey>
    public keystore value
  </directUpdateAuthenticityPublicKey
>
</application>
```

**Notes:**

- It is highly suggested to enable Secure Direct Update.
- Secure Direct Update does not work on already-deployed applications.
- Secure Direct Update works on applications published with the above configuration.

> For more information, see the topic about configuring and customizing direct update, in the user documentation.

## Sample application

Click to download (https://github.com/MobileFirst-Platform-Developer-Center/CustomDirectUpdate/tree/release71) the MobileFirst project.