Using Direct Update to quickly update your application

fork and edit tutorial (https://github.ibm.com/MFPSamples/DevCenter/tree/master/tutorials/en/foundation/6.3/advanced-client-side-development/using-direct-update-quickly-update-application.html) | report issue (https://github.ibm.com/MFPSamples/DevCenter/issues/new)

About Direct Update

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

The topics covered, are:

- advantages and restrictions of using Direct Update
- 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

Advantages

- Using Direct Update, organizations can ensure that users always use the latest version of the application
- Better control of application versions, by notifying users of pending updates or preventing the use of obsolete versions
- Updates that are deployed to the MobileFirst Server are automatically pushed to user devices
- Possible to push an update silently without user interaction (requires creating a custom Direct Update -- see below)

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 pre-defined authentication realm and challenge handler, and can be adjusted in `authenticationConfig.xml` to the following modes:

- perSession

- perRequest
- disabled

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" />
```

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:

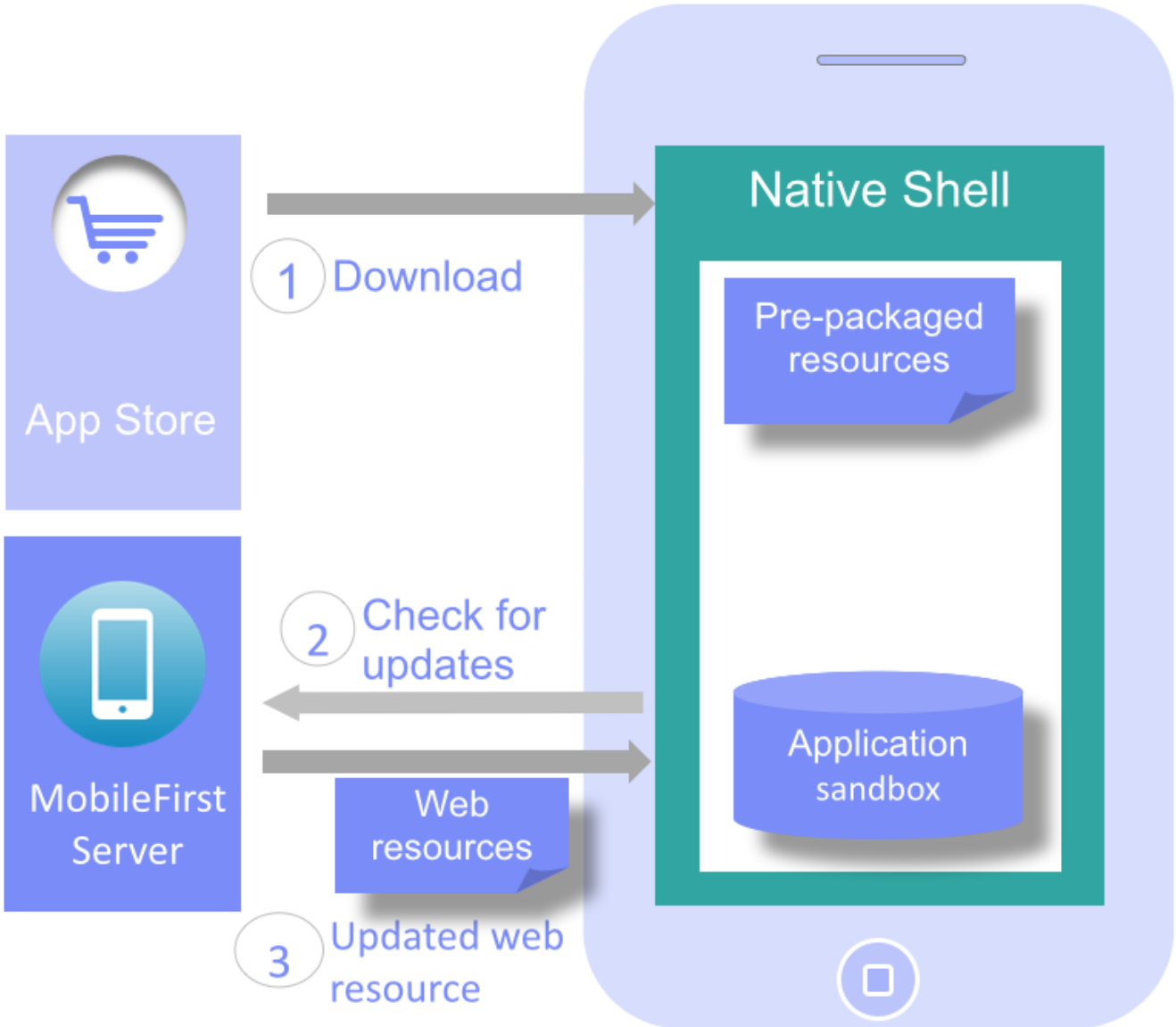
```
<customSecurityTest name="customTests">
   <test realm="wl_directUpdateRealm" mode="perRequest"/
>
</customSecurityTest>
```

For more information, review the “Direct Update as a security realm” user documentation topic.

Internal function

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

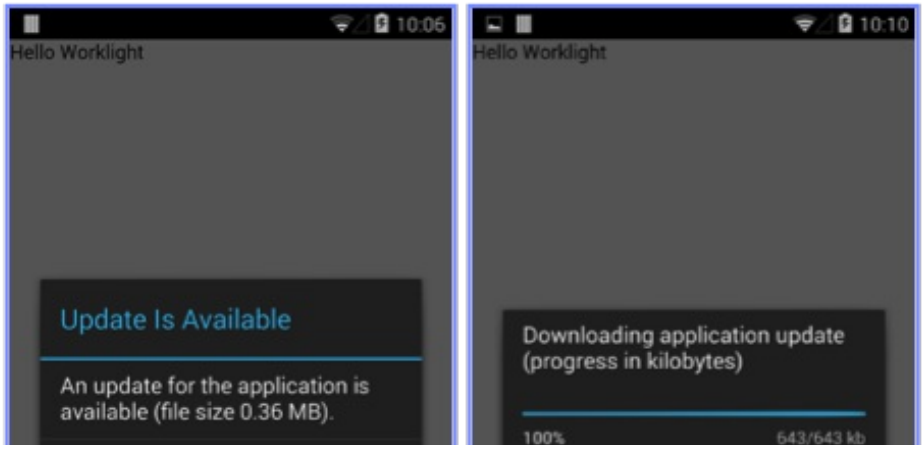
The updated web resources are downloaded when necessary.
After a Direct Update, the application will no longer use pre-packaged resources but those resources in the application's sandbox.

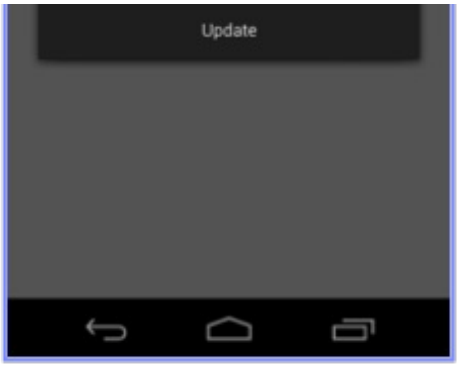


User experience

Default behavior

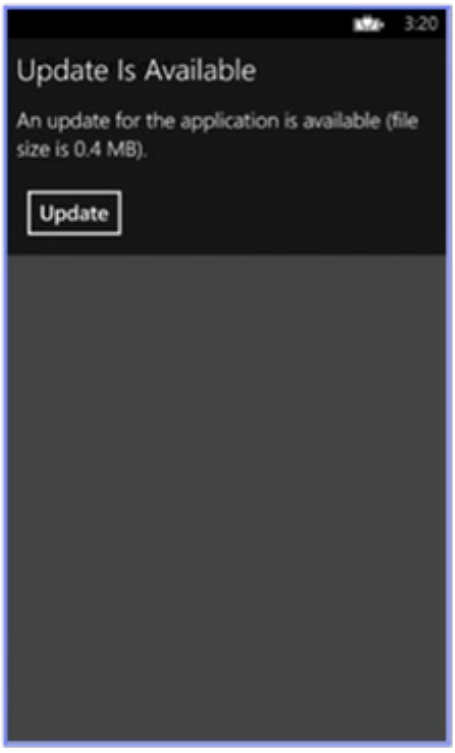
By default, once a Direct Update is received a pop-up dialog is displayed and the user is asked whether to begin the update process.
Once the user approves, a progress bar dialog is displayed and web resources are downloaded. The application is automatically reloaded once the update is complete.

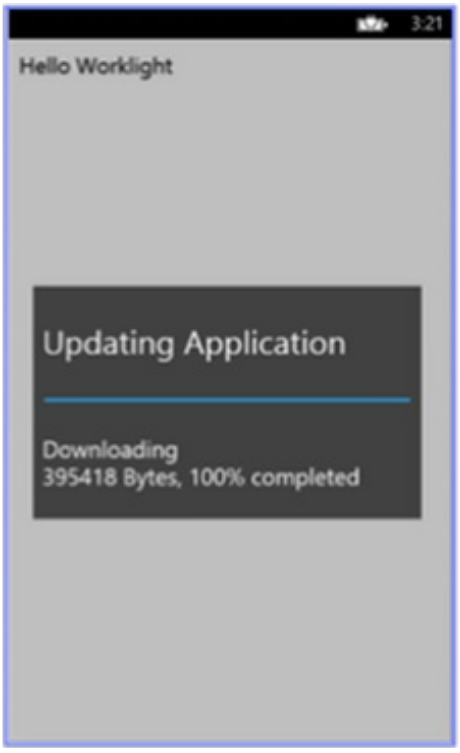



Update

GOLAN T
09:33
HelloWorklight
Update Is Available
An update for the application is available (file size 0.32 MB).
Update

GOLAN T
13:29
Downloading...


3:20
Update Is Available
An update for the application is available (file size is 0.4 MB).
Update


3:21
Hello Worklight
Updating Application
Downloading
395418 Bytes, 100% completed

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 will be downloaded and updated. This reduces download time, conserves bandwidth and improves overall user experience.

It is important to note that a differential update is only possible if the client application's web resources are one version behind the currently deployed application 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), will receive a full update, meaning the entire web resources will be downloaded and updated.

There is no change in the behaviour of applications 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) {<br />
    // custom Direct Update logic<br />
};
```

`directUpdateData` - A JSON object containing the `downloadSize` property that represents the files size in bytes of the update package to be downloaded from the MobileFirst Server.

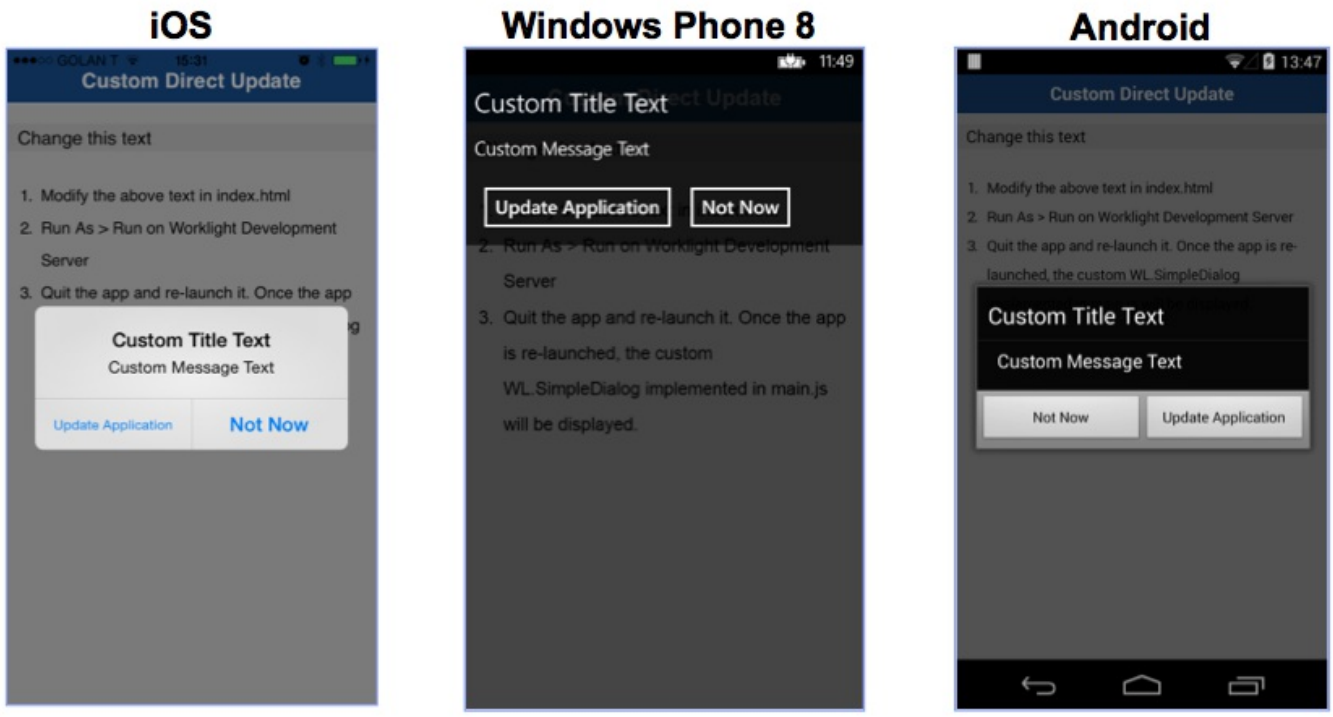
`directUpdateContext` - A JavaScript object exposing `.start()` and `.stop()` functions that start and stop the Direct Update flow.

Example

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

- 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';</p>
    WL.SimpleDialog.show(customDialogTitle, customDialogMessage,
        [{
            text : customButtonText1,
            handler : function() {
                directUpdateContext.start();
            }
        },
        {
            text : customButtonText2,
            handler : function() {
                wl_directUpdateChallengeHandler.submitFailure();
            }
        }]
    );
};
```



In the example code above, `submitFailure` was 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.

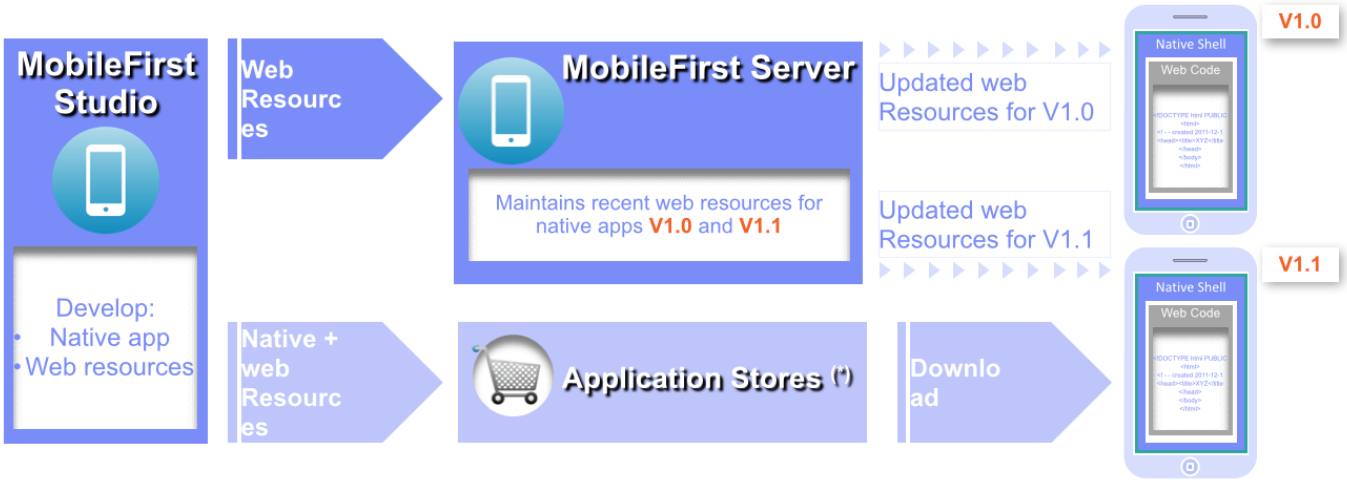
As such, it is important to call `submitFailure()` to notify the MobileFirst framework that the process completed with a “failure”. The MobileFirst framework in turn invokes the `onFailure` callback of the invocation that triggered the Direct Update.

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’s lifecycle. For more information, see the “Customizing the direct update interface” user documentation topic.

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

Distribution

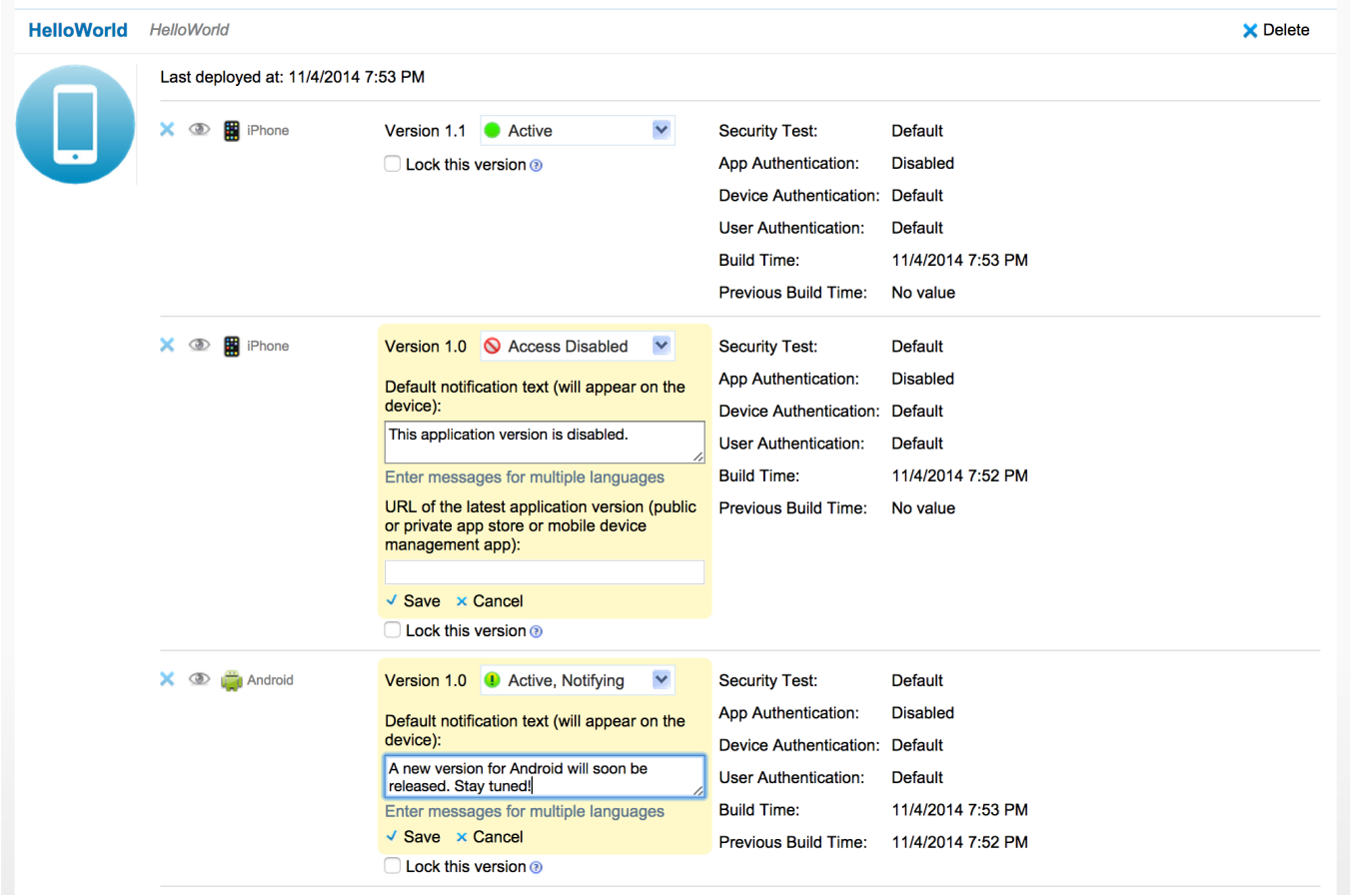


(*) During development cycles, testers automatically get recent web resources through internal distribution mechanisms and not through application stores.

Disabling old application versions

From the MobileFirst 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 the 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

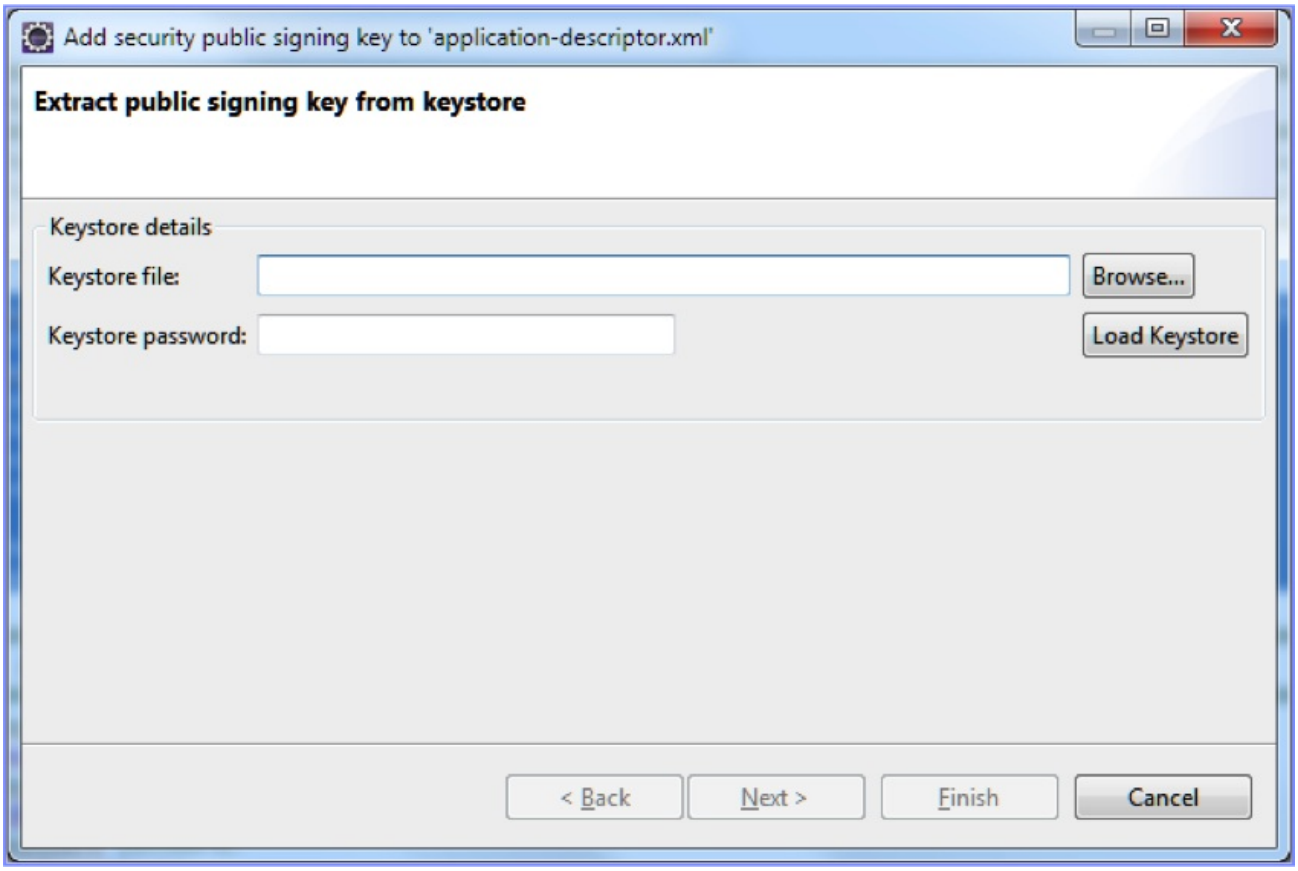
Enabling Direct Update authenticity prevents altering the transmitted web resources from the server to the client application by a 3rd party attacker (i.e., when it is cached in a CDN).

To enable Direct Update authenticity:

1. Generate a certificate and place it in the MobileFirst project under the `server\conf folder`
2. Edit the MobileFirst Default Certificate section in `server\conf\worklight.properties` with the certificate's 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's public key using the Application Descriptor Editor view. To do so:
   - Right-click on the application folder and select Extract Public Signing Key
   - Follow the on-screen instructions:



You can find the public key on the Design view of the Application Descriptor Editor.

You can also find the public key in the Source view of the Application Descriptor Editor.

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

</directUpdateAuthenticityPublicKey>
</application>
```

Notes:

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

> For more information, review the "Configuring and customizing direct update" user documentation topic.

Sample application

Click to download (http://public.dhe.ibm.com/software/products/en/MobileFirstPlatform/docs/v630/CustomDirectUpdateProject.zip) the Studio project.