Using Direct Update in Cordova applications

TODO: update images

Overview

With Direct Update, Cordova applications can be updated "over-the-air" with refreshed web resources, such as changed, fixed or new applicative logic (JavaScript), HTML, CSS or images. Organizations are thus able to ensure that end-users always use the latest version of the application.

In order to update an application, the updated web resources of the application need to be packaged and uploaded to the MobileFirst Server using the MobileFirst CLI, which will then handle updating any application as it attempts to connect.

Supported Cordova platforms: iOS and Android.

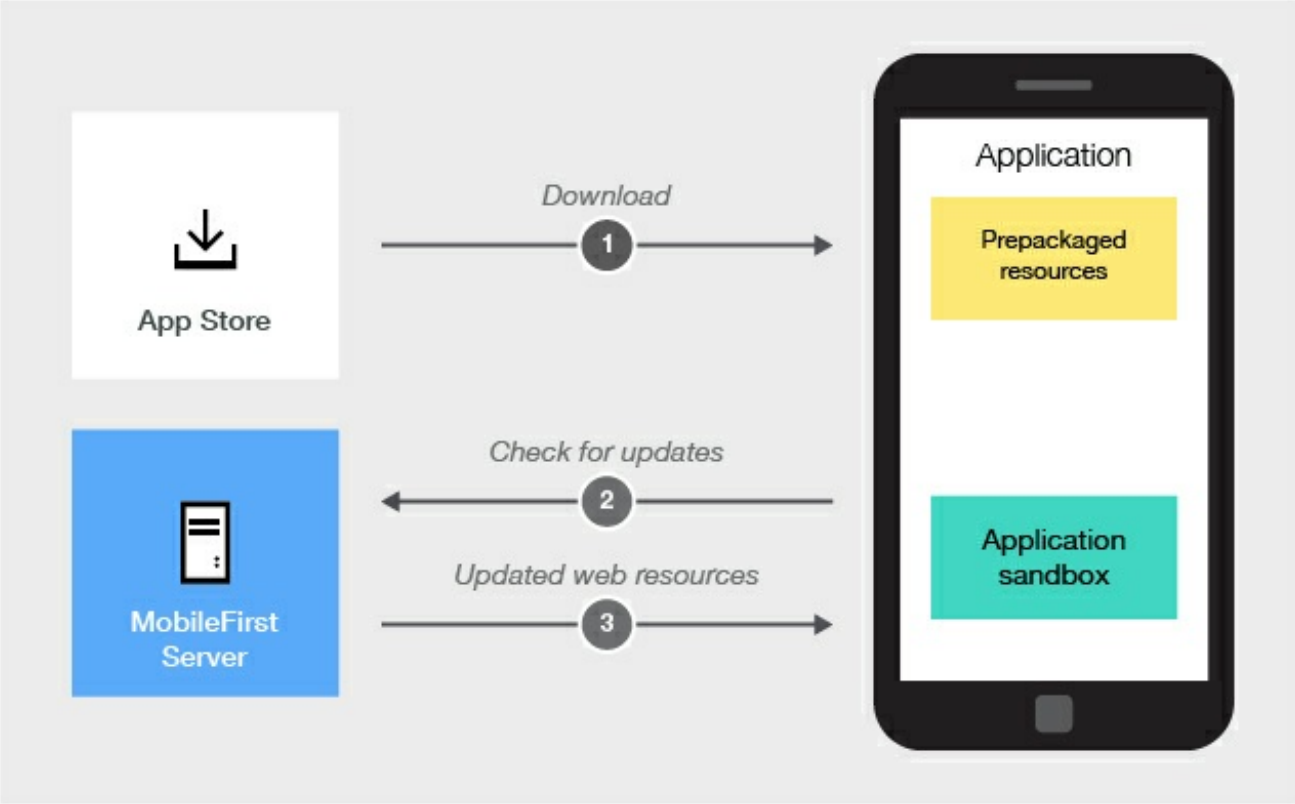
> Direct Update updates only the application's web resources. To update native resources a new application version must be submitted to the respective app store.

Jump to:

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 on every request to the MobileFirst. The updated web resources are downloaded when necessary.

After a Direct Update, the application no longer uses the pre-packaged web resources. Instead it will use the web resources in the application's sandbox.



Creating and deploying updated web resources

Once work on new web resources, such as bug fixes or minor changes and the like, is done, the updated web resources need to be packaged and uploaded to the MobileFirst Server.

1. Open a **Command-line** window and navigate to the root of the Cordova project.
2. Run the command: `mfpdev app webupdate`.

The `mfpdev app webupdate` command packages the updated web resources to a .zip file and uploads it to the default MobileFirst Server running in the developer workstation. The packaged web resources can be found at the **[cordova-project-root-folder]/mobilefirst/** folder.

Alternatives:

- Build the .zip file but do not upload it:

```
mfpdev app webupdate --build
```

- Build the .zip file and upload it to a different MobileFirst Server: `mfpdev app webupdate [server-name] [runtime-name]`. For example:

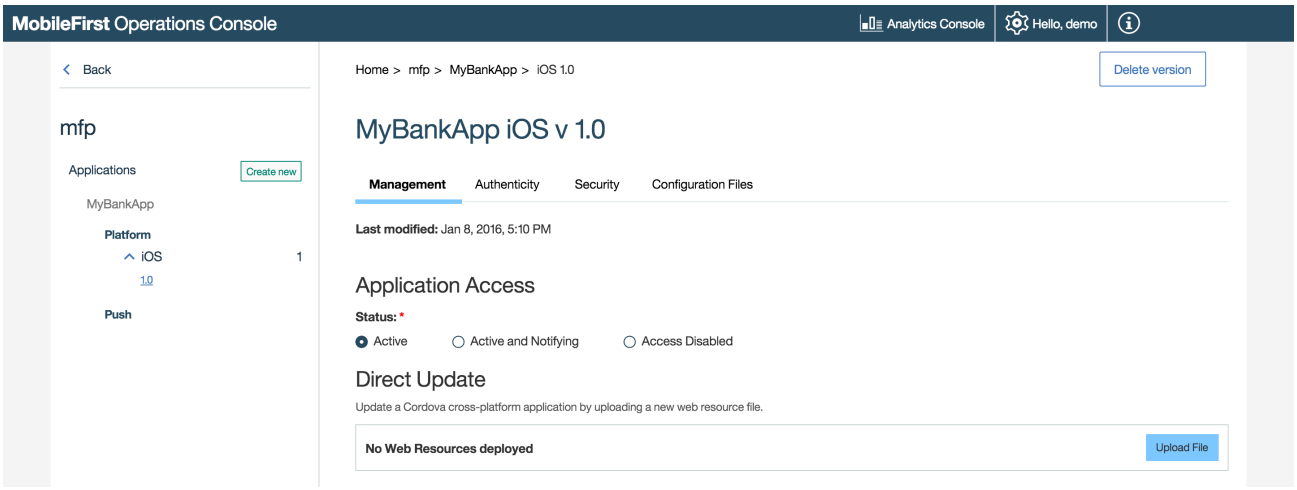
```
mfpdev app webupdate myQAServer MyBankApps
```

- Build the .zip file and upload a previously generated .zip file: `mfpdev app webupdate [server-name] [runtime-name] --file [path-to-packaged-web-resources]`. For example:

```
mfpdev app webupdate myQAServer MyBankApps --file mobilefirst/ios/com.mfp.myBankApp-1.0.1.zip
```

- Build the .zip file and upload it via the MobileFirst Operations Console:

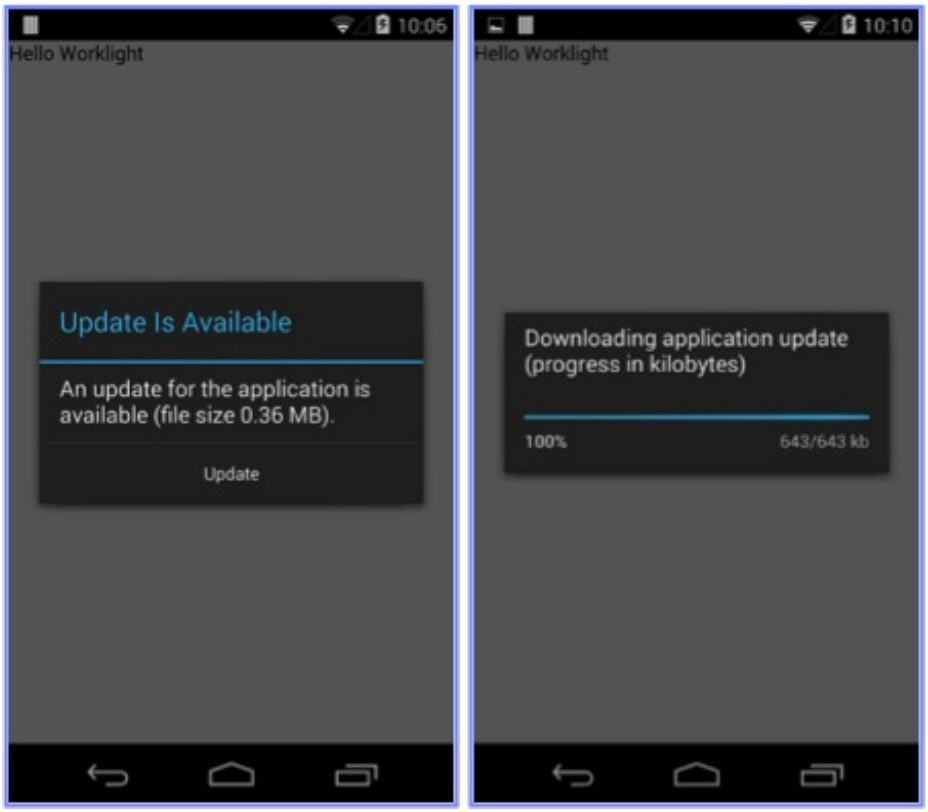
```
mfpdev app webupdate --build
```

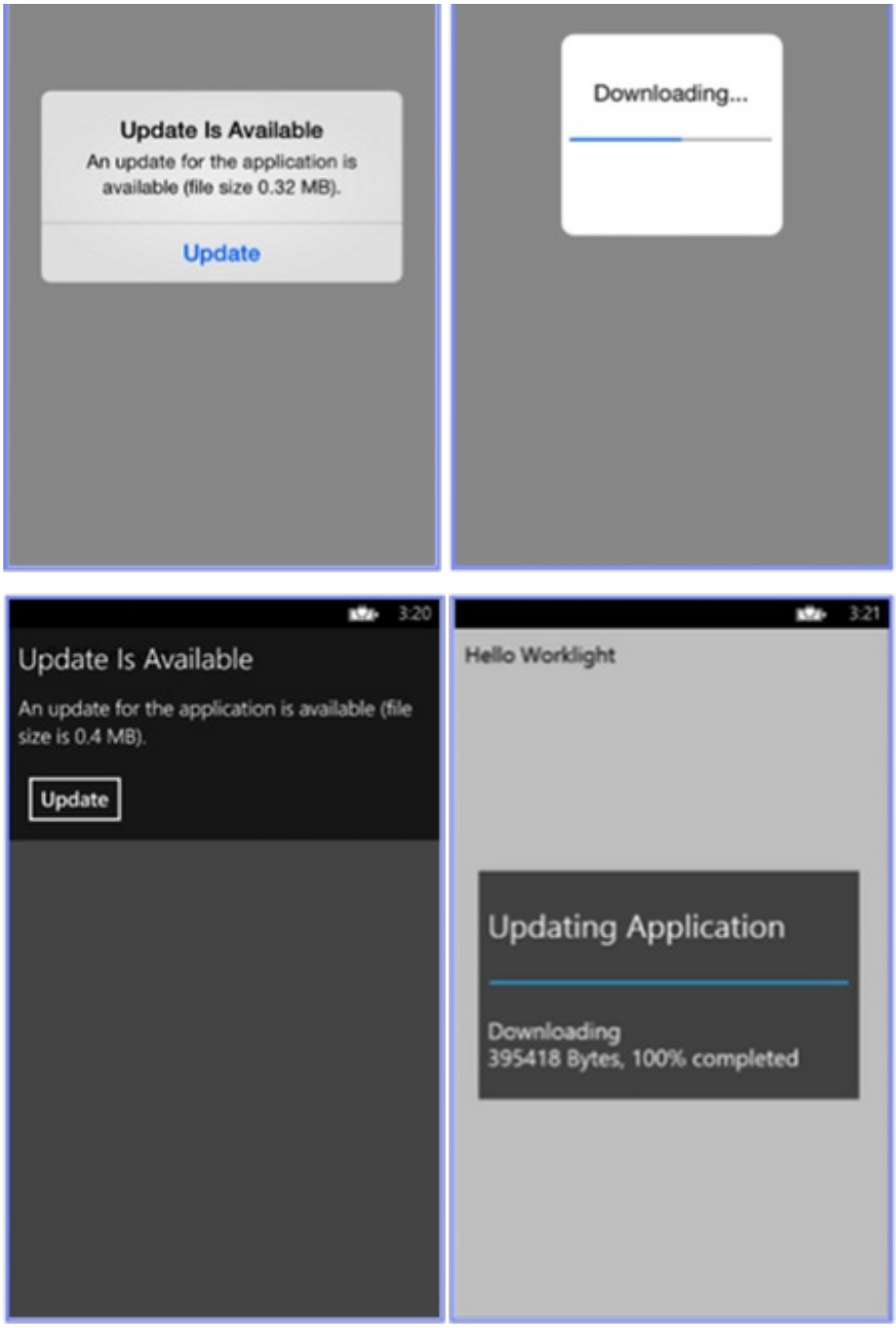


> Run the command `mfpdev help webupdate` to learn about additional command flags.

User Experience

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





Customizing the Direct Update UI

The default Direct Update UI that is presented to the end-user can be customized, as can the entire user experience flow.
To do so, override the `handleDirectUpdate` function:

```
wl_DirectUpdateChallengeHandler.handleDirectUpdate = function(directUpdateData, directUpdateContext)
{
   // Implement 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 to the user to either continue with the update process or dismiss it.
Additional examples for a customized Direct Update UI:

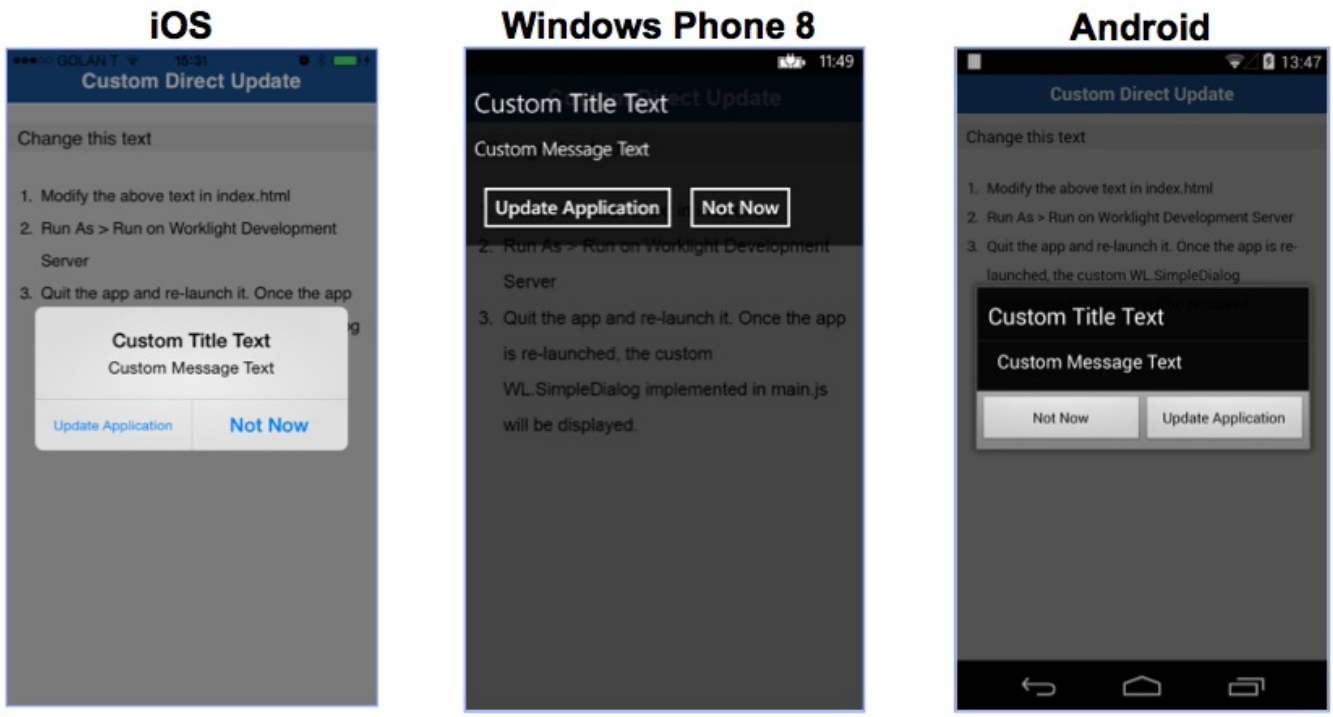
- A dialog that is created by using a third-party JavaScript framework (such as Dojo or jQuery Mobile, Ionic, ...)
- Fully native UI by executing a Cordova plug-in
- An alternate HTML that is presented to the user with options
- And so on…

```
wl_directUpdateChallengeHandler.handleDirectUpdate = function(directUpdateData, directUpdateContext)
{
    // custom text for the dialog
    var customDialogTitle = 'Custom Title Text';
    var customDialogMessage = 'Custom Message Text';
    var customButtonText1 = 'Update Application';
    var customButtonText2 = 'Not Now';

    // Create dialog
    navigator.notification.confirm(
        'Custom Message Text',
         onClick,
        'Custom Title Text',
        ['Update','Cancel']
    );
};

// Handle dialog buttons
function onClick(buttonIndex) {
    if (buttonIndex == 1) {
        directUpdateContext.start();
    } else {
        wl_directUpdateChallengeHandler.submitFailure();
    }
}
```

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



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.

Further customization

Optionally, a developer can also supply a Direct Update listener to fully control a Direct Update lifecycle entirely.

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

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

Direct Update authenticity

Direct Update authenticity prevents a 3rd-party attacker from altering the web resources that are transmitted from the MobileFirst Server (or from a content delivery network (CDN)) to the client application.

> Note: Direct Update authenticity is disabled by default. To enable, see the "Configuring and customizing direct update" user documentation topic.

Differential Direct Update

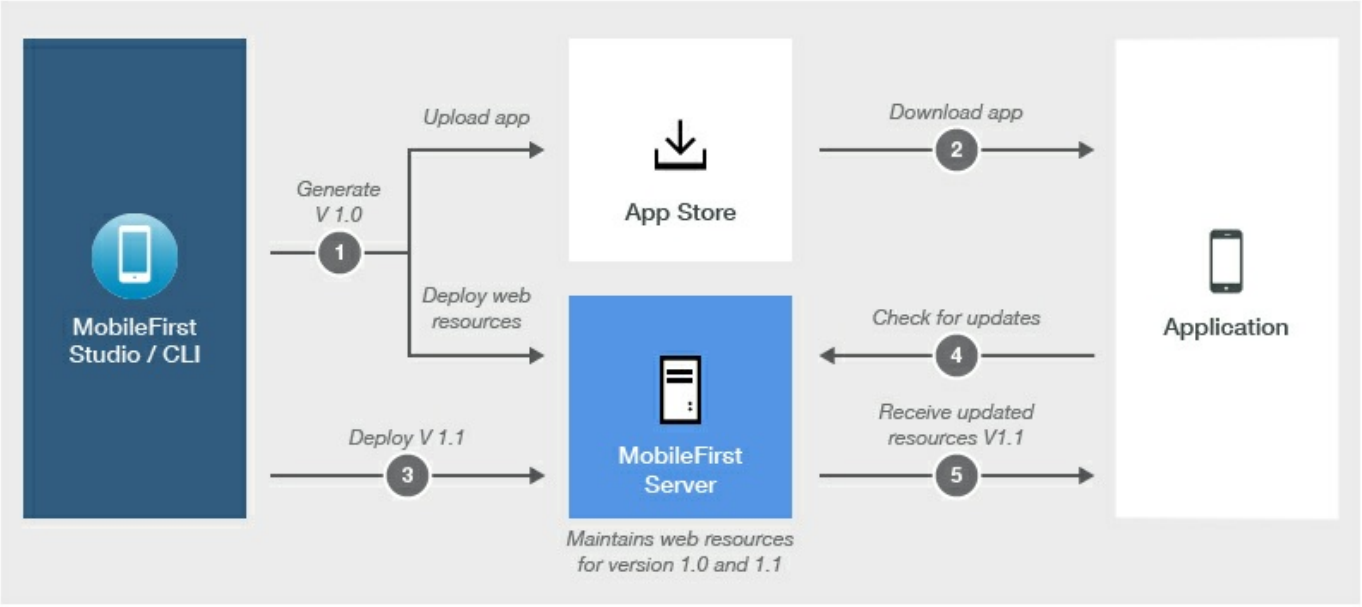
Differential Direct Updates enables an application to download only the files that were changed since the last update instead of the entire web resources of the application. This reduces download time, conserves bandwidth, and improves overall user experience.

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 currently deployed application (meaning 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.

Working with Direct Update in the field

The diagram below depicts the flow of updating an application's web resources using Direct Update once it has been submitted to the application stores and used by end-users.



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

Sample application

Click to download (https://github.com/MobileFirst-Platform-Developer-Center/CustomDirectUpdate/tree/release80) the Cordova project.

Sample usage

1. From the command line, navigate to the Cordova project.
2. Add a platform using the `cordova platform add` command.
3. Prepare and run the Cordova application using `cordova prepare` followed by `cordova run`.
4. Alter the application web resources and upload them to the MobileFirst Server as explained above.